TAMPEREEN
AMMATTIKORKEAKOULU

PROCESSWIRE-JÄRJESTELMÄN PERUSTEET KEHITTÄJILLE

Teppo Koivula

Opinnäytetyö
Joulukuu 2015
Tietojärjestelmäosaamisen koulutusohjelma, YAMK

TIIVISTELMÄ

Tampereen ammattikorkeakoulu
Tietojärjestelmäosaamisen koulutusohjelma, YAMK

KOIVULA, TEPPO:
ProcessWire-järjestelmän perusteet kehittäjille

Opinnäytetyö 120 sivua, joista liitteitä 96 sivua
Joulukuu 2015

Tämän opinnäytetyön tavoitteena oli tuottaa monipuolinen, helppokäyttöinen ja ennen kaikkea suomenkielinen perehdytysmateriaali sivustojen, sovellusten ja muiden web-ympäristössä toimivien ratkaisujen toteuttamiseen hyödyntäen sisällönhallintajärjestelmää ja sisällönhallintakehystä nimeltä ProcessWire.

ProcessWire on avoimen lähdekoodin alusta, jonka suunniteltu käyttöympäristö on PHP-kielen, MySQL-tietokannan sekä Apache-web-palvelimen muodostama palvelinympäristö. Koska järjestelmä sisältää piirteitä sekä sisällönhallintajärjestelmistä että sisällönhallintakehyksistä, se on käytännössä osoittautunut erittäin joustavaksi ratkaisuksi monenlaisiin web-pohjaisiin projekteihin.

Opinnäytteen varsinaisena lopputuotteena syntyi opas, jonka tavoitteena on sekä teoriapohjan että käytännön ohjeistuksen välittäminen perustuen todellisiin projekteihin ja niiden tiimoilta esiin nousseisiin havaintoihin. Paitsi perehdytysmateriaalina järjestelmään tutustuville uusille käyttäjille, oppaan on jatkossa tarkoitus toimia myös kokeneempien käyttäjien apuvälineenä.

Opinnäytetyöraportin ensimmäinen luku perehdyttää lukijan verkkopalvelujen teknisiin alustaratkaisuihin pääpiirteissään, minkä jälkeen käydään opinnäytteen lopputuotteena syntyneen oppaan luvut yksi kerrallaan läpi ja kuvataan kunkin osalta niin sen tavoite ja tarkoitus, kuin myös kyseisen oppaan luvun pohjalta tehdyt johtopäätökset. Varsinainen opas on kokonaisuudessaan opinnäytteen liitteenä.

Asiasanat: sisällönhallintajärjestelmä, sisällönhallintakehys, sovelluskehitys, käsikirja

ABSTRACT

Tampereen ammattikorkeakoulu
Tampere University of Applied Sciences
Master's Degree Programme in Information System Competence

KOIVULA, TEPPO:
Developer's Guide for ProcessWire CMS/CMF

Master’s Thesis 120 pages, appendices 96 pages
December 2015

The purpose of this Master's Thesis was to create a comprehensive, easy-to-use guidebook in Finnish for developing websites, web applications, and other forms of web-based solutions with the content management framework (CMF) and content management system (CMS) ProcessWire.

ProcessWire is an open source platform designed to run natively on a combination of PHP, MySQL, and the Apache web server. The system includes features of CMS and CMF platforms, and has already proven to have the flexibility to power just about all types and sizes of web based solutions.

The end product of this thesis is a guidebook aiming to provide the reader with both the necessary theoretical background and useful and practical examples based on real world applications. Primary use cases for this guidebook include introducing new users to the platform and providing more experienced users with an easy to browse reference guide.

The thesis starts from theory by introducing the reader to a variety of web publishing tools available for modern web-based sites and application. From there on it continues by walking the reader through each specific chapter of the guidebook itself, explaining and summarizing each of them one by one.

Keywords: content management system, content management framework, software development, guidebook

SISÄLLYS

1 JOHDANTO

Suomalaiset web-alan yritykset ovat perinteisesti nojanneet itse kehittämiinsä ratkaisuihin, joista markkinoilla näkyvät vahvasti esimerkiksi Koodiviidakon Sivuviidakko, Poutapilven P4 ja Maisema, Crasmanin Stage, sekä nykyisin Fonectan omistama Kotisivukone. Näiden rinnalle ovat etenkin viimeisten vuosien aikana – myös isompien toimijoiden parissa – nousseet vahvasti avoimen lähdekoodin järjestelmät, joista suosituimpiin lukeutuvat esimerkiksi WordPress, Drupal, Liferay, ja Concrete5.

Vuonna 2011 työskennellessäni Optinet Oy:n liiketoimintayksikössä Aldonessa toteutimme ensimmäiset asiakasprojektimme hyödyntäen tuolloin vielä hyvinkin tuntematonta järjestelmää nimeltä ProcessWire. Tätä ennen olimme jo pitkään hyödyntäneet kaikissa projekteissamme yrityksen sisäisenä kehityksenä toteutettua julkaisujärjestelmää Directoa, jonka rinnalle ProcessWire aluksi tuotiinkin vain uutena vaihtoehtoisena, avoimeen lähdekoodiin perustuvana ratkaisuna.

Näiden aivan ensimmäisten toteutusten jälkeen olen itse ollut mukana rakentamassa useita kymmeniä sivustoja, sovelluksia, ja erilaisia portaalimaisia toteutuksia ProcessWire-alustalle, sekä nähnyt omin silmin järjestelmän käyttäjämäärien ja suosion vuosi vuodelta kasvavan. Aktiivinen toiminta ProcessWiren kehittäjäyhteisössä on avartanut näköalaa, ja tätä kautta on myös hankittu suuri osa siitä osaamisesta ja ymmärryksestä, joka allekirjoittaneelta kyseisen järjestelmän osalta löytyy.

Suomessa on tällä hetkellä kourallinen yrityksiä ja yksittäisiä freelancer-kehittäjiä, jotka hyödyntävät juuri ProcessWirea ensisijaisena työkalunaan. Vaikka määrä ei ole vielä mitenkään päätähuimaava, järjestelmä on kuitenkin löytänyt oman, uskollisen käyttäjäkuntansa etenkin niiden kehittäjien ja yritysten parista, joilla on aiempaa kokemusta tiettyjen kilpailevien järjestelmien kankeudesta, monimutkaisuudesta, ja toistuvista ongelmista etenkin tietoturvan suhteen.

Tammikuussa 2013 Aldone siirtyi Fonecta Oy:n omistukseen, mutta varsinainen tekeminen säilyi kuitenkin – etenkin järjestelmien osalta – pitkälti ennallaan. Nykyisin entinen Aldonen tiimi on jo sulautunut osaksi Fonectan räätälöityjen ratkaisujen tuotantoa, jossa muiden alan asiantuntijoiden mukana olemme tarjoamassa palveluja yritysten, yhteisöjen, ja julkishallinnon digitaaliseen myyntiin, markkinointiin, ja verkkoasiointiin.

1.1 Opinnäytetyön tavoite ja tarkoitus

ProcessWire on etenkin suomalaisten web-kehittäjien parissa suhteellisen tuntematon järjestelmä, joten uusien kehittäjien perehdyttämiseen on jo jonkin aikaa kaivattu kunnollista opiskelu- ja referenssimateriaalia. Ilman hyvää opiskelumateriaalia uusien käyttäjien tuominen järjestelmän pariin on haastavaa, ja suppea käyttäjämäärä puolestaan vaikuttaa suoraan – ja negatiivisesti – myös itse järjestelmän arvostukseen.

Koska perehtyminen laajan järjestelmän toimintaan ilman hyvää lähdemateriaalia on kohtuuttoman työlästä ja hidastaa järjestelmän omaksumista aivan tarpeettomasti, joten hyvälle suomenkieliselle ohje- ja opastusmateriaalille koettiin olevan tilausta. Englanninkielistä materiaalia toki on saatavilla, mutta sekin on toistaiseksi varsin hajanaista ja vaatii siten melkoisesti syventymistä.

Tämän opinnäytetyön ensisijainen tavoite onkin järjestelmän virallisen, englanninkielisen ohjemateriaalin täydentäminen erityisesti aloittelevan käyttäjän tarpeet huomioivan suomenkielisellä oppaan avulla. Varsinainen oppaan osuus on opinnäyteraportin liitteenä, ja raportissa puolestaan keskitytään dokumentoimaan opastekstin syntyä ja siinä käsiteltyjä asioita pääpiirteissään.

1.2 Opinnäytetyön rakenne

Koska riittävän teoria- ja taustatiedon merkitystä ei voi liiaksi korostaa, opinnäytteen alkupuoli keskittyy verkkopalvelujen alustaratkaisuihin yleisellä tasolla. Tarkoituksena on tutustuttaa lukija siihen ympäristöön, jossa web-kehittäjät toimivat, sekä tarjota perustason ymmärrys liittyen kehittäjien käyttöön suunnattuihin teknisiin ratkaisuihin.

Seuraavien lukujen osalta opinnäytteen rakenne noudattaa oppaan rakennetta. Kukin opinnäytteen luku esittelee oppaan vastaavan luvun tavoitteet ja siinä käsitellyt asiat pääpiirteissään. Lisäksi opinnäytteen luvut sisältävät jonkin verran oppaassa käsiteltyihin asioihin liittyvää pohdintaa sekä näiden pohjalta vedettyjä johtopäätöksiä.

Opinnäytteellä ei ole omaa erityissanastoaan, sillä vastaava luettelo löytyy jo oppaan lopusta otsikon ”lyhenteet ja sanasto” alta.

1.3 Oppaan kohderyhmä ja rakenne

Opinnäytteen liitteistä löytyvä ProcessWire-kehittäjän opas on laadittu ensisijaisesti aloitteleville ProcessWire-kehittäjille, eikä se edellytä lukijalta aiempaa kokemusta järjestelmästä. ProcessWire-kehityksen näkökulmasta vähintään perustason ymmärrys web-sivustojen koostamisesta on kuitenkin suositeltavaa, eikä PHP-kielen – tai muiden korkean tason ohjelmointikielten – tuntemuksestakaan varsinaisesti haittaa ole.

Oppaan rakenne mukailee tyypillisen ohje- tai opaskirjan rakennetta, ja lähestymistapa on suoraviivainen ja esimerkkeihin nojaava. Ensimmäinen luku esittelee ProcessWire-järjestelmän suunnittelu- ja arkkitehtuuriperiaatteita, toinen luku tietorakenteiden muodostamista, ja myöhemmissä luvuissa käsitellään muun muassa sivuaihioiden toteuttamista, välimuistiratkaisujen hyödyntämistä, sekä moduulikehityksen perusteita.

Sivuaihiot edustavat ProcessWire-sivuston näkyvää osuutta, ja niitä käsitellään oppaassa itse asiassa kokonaisten kahden luvun verran: oppaan kolmannessa luvussa käydään ensin läpi sivuaihioiden toteuttamisen perusteet varsin yleisellä tasolla, ja tämän jälkeen neljännessä luvussa esitellään erilaisia aihiotason rakennemalleja, mukaan lukien kevyt, Model-View-Controller -arkkitehtuuriin pohjautuva malli.

1.4 Oppaan sisällön rajaus

Oppaassa käsitellään ProcessWire-järjestelmää varsin laajasti, mutta joitakin rajauksia sen sisältöön on käytännön syistä jouduttu tekemään. Esimerkiksi järjestelmän hyödyntäminen bootstrap-menetelmän avulla puhtaasti palvelinpäässä, web-kontekstin ulkopuolella, on tästä oppaan versiosta jätetty kokonaisuudessaan pois.

Opas ei myöskään käsittele yleisesti web-kehitykseen tai PHP-kieleen liittyviä aihepiirejä, kuten esimerkiksi kehitys- tai tuotantoympäristön käyttöönottoa, ympäristön ja yksittäisen sovelluksen suorituskyvyn optimointia, saati edes tietoturvaa. Näihin liittyen löytyy jo muutoinkin huomattava määrä erittäin tasokasta opiskelumateriaalia.

2 VERKKOPALVELUJEN ALUSTARATKAISUT LYHYESTI

Internetin varhaisina vuosina oli täysin luonnollista ylläpitää sivustoja manuaalisesti, laatimalla yksilöllisiä HTML-tiedostoja käsin. Laajojen sisältöjen hallinnointi tällä tavoin ei kuitenkaan ole erityisen kustannustehokasta, suurempien muutosten tekeminen jälkikäteen on varsin työlästä, ja HTML-tiedostojen tuottaminen käsin vaati osaamista, jonka välittämisessä kaikille sisällöntuotantoon osallistuville on omat ongelmansa.

Jo ensimmäisen aallon sisällönhallintajärjestelmät toivat mukanaan uudenlaisen, suoraviivaisemman, ja monin tavoin tehokkaamman tavan hallinnoida sisältöä verkossa. Vasta WYSIWYG-työkalujen yleistymisen myötä verkkosisällön tuottamisen voitiin kuitenkin katsoa yksinkertaistuneen siinä määrin, ettei se enää edellyttänyt merkittävää erityisosaamista, saati teknistä tuntemusta.

Nykyisin tyypillisen sisällönhallintajärjestelmän käyttöliittymä jäljittelee graafisen tuotannon työvälineiden, taitto-ohjelmien, sekä toimisto-ohjelmien käyttöliittymiä, jotka ovat monelle jo entuudestaan tuttuja. Tähän tosin loppuvatkin käyttöliittymätason yhteneväisyydet sisällönhallintajärjestelmien välillä, sillä juuri hallintatyökalut ovat niille varsin ominainen erottautumiskeino.

Teknisestä näkökulmasta katsottuna eräs sisällönhallintajärjestelmien tärkeimmistä eduista on esitystavan, rakenteen, ja sisällön erottaminen toisistaan. Tämä tekee mahdolliseksi yhden osa-alueen muuttamisen muiden säilyessä ennallaan. Moni järjestelmä mahdollistaa myös sisällön esittämisen eri tavoin riippuen esimerkiksi siitä, millä tavoin, mistä maantieteellisestä sijainnista, millaisilla käyttäjäkohtaisilla asetuksilla, tai minkälaisella laitteella sitä tarkastellaan.

2.1 Sisällönhallintajärjestelmät, sovelluskehykset ja sisällönhallintakehykset

Verkkopohjaisten ratkaisujen toteuttamiseen on käytettävissä valikoima erilaisia työkaluja, joita käsitellään tässä opinnäytteessä ymmärrettävyyden vuoksi kolmena pääryhmänä: sisällönhallintajärjestelmät, sovelluskehykset ja sisällönhallintakehykset. Luokittelu ei ole aukoton ja yksi järjestelmä voi kuulua samanaikaisesti useampaankin ryhmään, mutta näin kokonaisuuden hahmottaminen on hieman helpompaa.

Tässä luvussa perehdymme ensisijaisesti mainittujen pääryhmien ominaisuuksiin ja erityispiirteisiin. Tuotekohtaiseen vertailuun emme juuri käytä aikaa, sillä huomattavasti tärkeämpää on tiedostaa tuotevalikoiman laajuus sekä se, että tuotteiden asettaminen keskinäiseen paremmuusjärjestykseen harvoin tuottaa mielekästä lopputulosta. Teknisen alustan valinta on vaativa prosessi jopa tähän tehtävään erikoistuneille asiantuntijoille.

2.2 Sisällönhallintajärjestelmät

Sivustotuotannon ja verkkopalvelujen näkökulmasta sisällönhallintajärjestelmä (CMS tai WCMS) on tyypillisesti selainkäyttöinen, verkossa toimiva sovellusohjelma, jonka avulla loppukäyttäjät voivat hallita sivuston sisältöä: lisätä uusia sivuja, muokata sivuilta löytyvää sisältöä, ja poistaa tarpeettomaksi käyneitä sivuja ja sisältöjä. Koska sisällönhallintajärjestelmien kirjo on laaja, myös niiden toiminnallisuudet ja ominaispiirteet vaihtelevat merkittävästi järjestelmästä toiseen.

Moni sisällönhallintajärjestelmä tarjoaa valmiiden sisällönhallintatyökalujen lisäksi työvälineitä erityisesti kehittäjille. Myös näiden luonne vaihtelee: toisessa ääripäässä ovat pitkälle tuotteistetut itsepalvelujärjestelmät, jotka korkeintaan mahdollistavat pienehköjen visuaalisten muutosten tekemisen verkkopohjaisen käyttöliittymän avulla, kun taas toinen ääripää ovat järjestelmät, jotka mahdollistavat käytännössä kaikkien ratkaisun osa-alueiden täydellisen muokkaamisen tarpeen mukaan.

Monet nykyaikaiset julkaisujärjestelmät ovat laajennettavissa lisäosilla, jolloin järjestelmän toimintojen räätälöinti onnistuu jopa ilman sovellustason muutoksia. Tiettyihin järjestelmiin on saatavilla suuri määrä lisäosia, toisiin taas varsin rajallinen määrä. Lisäosien valinnassa on kuitenkin syytä olla valveutunut, sillä tyypillisesti lisäosilla on laaja pääsy järjestelmään ja sen hallinnoimiin tietoihin, jolloin haavoittuva lisäosa saattaa helposti koko järjestelmän alttiiksi hyökkäyksille.

Muutamia esimerkkejä tyypillisistä sisällönhallintajärjestelmistä:

- **Drupal** on sisällönhallintajärjestelmä, joka mahdollistaa erittäin laajan ja monipuolisen räätälöinnin niin näkymien, hallintatoimintojen, kuin tietotyyppienkin osalta. Toisaalta Drupal tunnetaan myös monimutkaisena ja työläänä järjestelmänä, jonka oppiminen ottaa aina oman aikana. (Mitchell 2013).

- **WordPress** oli alkujaan blogityökalu, joka on vuosien varrella kasvanut ja kehittynyt merkittävästi. Nykyisin WordPress on yksi suosituimmista julkaisujärjestelmistä, sillä väitetysti jopa 24% kaikista tunnetuista verkkosivustoista on toteutettu sen avulla. WordPressille on myös saatavilla erittäin laaja valikoima valmiita ulkoasuteemoja ja lisäsovelluksia. (WordPress.org 2015.)
- **Joomla!** on puhtaasti yhteisön tukema julkaisujärjestelmä, joka nykymuodossaan käsittää niin laajat sisällönhallintatyökalut kuin MVC-arkkitehtuuriin perustuvan sovelluskehyksenkin (Severdia 2015).

2.3 Sovelluskehykset

Sovelluskehys viittaa tässä yhteydessä tarkemmin web-sovelluskehykseen (WAF). Hyvin yleisellä tasolla sovelluskehys voidaan määritellä kokoelmaksi komponentteja, joita yhdistelemällä voidaan toteuttaa kokonaisia sovelluksia. Sovelluskehyksiä on kuitenkin olemassa varsin montaa erilaista tyyppiä, jotka eroavat niin ominaisuuksiltaan, toimintatavoiltaan, kuin laajuutensakin osalta.

Osa sovelluskehyksistä mahdollistaa yksittäisten komponenttien hyödyntämisen irrallisina paloina myös muiden ohjelmien sisällä, toisissa taas lähtökohtana on sovellusten rakentaminen kehyksen oman ytimen ympärille. Vastaavasti tietyt sovelluskehykset sisältävät laajan kattauksen erilaisia komponentteja (full-stack framework), kun taas toiset ovat todella minimaalisia (microframework).

Pääsääntöisesti sovelluskehykset ovat kuitenkin julkaisujärjestelmiin verrattuna huomattavasti matalamman tason työvälineitä, eivätkä sisällä esimerkiksi valmiita graafisia ylläpitonäkymiä laisinkaan. Sovelluskehykset soveltuvat ensisijaisesti projekteihin, joissa tavoiteltuna lopputuloksena on täysin räätälöity järjestelmä – tai esimerkiksi sovellus, joka ei yksinkertaisesti tarvitse ylläpitonäkymiä.

Muutamia esimerkkejä sovelluskehyksistä:

- **Zend Framework** on PHP-kielellä toteutettu avoimen lähdekoodin sovelluskehys, jonka tärkeimpänä tukijana toimii PHP:n taustaorganisaatio Zend Technologies. Zend Frameworkin kehitykseen ovat omalta osaltaan osallistuneet myös esimerkiksi Google ja Microsoft. (Zend Technologies 2015.)

- **CakePHP**, jota ylläpitää Cake Software Foundation, on eräs suosituimmista PHP-pohjaisista avoimen lähdekoodin sovelluskehyksistä. CakePHP on täysiverinen sovelluskehys, joka sisältää muun muassa ORM-ratkaisun ja suuren määrän muita komponentteja. (Cake Software Foundation 2015: 1-4.)
- **Phalcon** on PHP:n laajennuksena, eli C-kielellä, toteutettu sovelluskehys, jonka suunnittelussa ja toteutuksessa on annettu merkittävästi painoarvoa kehyksen nopealle toiminnalle ja suorituskyvylle. Phalconin hyödyntäminen ei edellytä kehittäjältä C-kielen tuntemusta. (Phalcon Team 2015.)

2.4 Sisällönhallintakehykset

Sisällönhallintakehys (CMF) on sisällönhallintajärjestelmän ja sovelluskehyksen välimuoto, joka sovelluskehyksen tapaan sopii räätälöityjen ratkaisujen toteuttamiseen ja sisällönhallintajärjestelmän tapaan tarjoaa valmiit työkalut sisällönhallintaan. Useimmissa sisällönhallintakehyksissä on graafinen ylläpitonäkymä, mutta sen käyttö on vapaaehtoista, sillä sisällönhallinta on mahdollista myös kehittäjärajapinnan ylitse.

Sisällönhallintakehykset ovat parhaimmillaan kun kaivataan räätälöityä ratkaisua, jonka toimintaan liittyy olennaisesti erilaisten tietotyyppien sekä mahdollisesti suuriksi kasvavien tietomassojen hallinnointi. Tällöin sisällönhallintakehyksen avulla voidaan välttää valmiin julkaisujärjestelmän taivuttaminen uuteen uskoon tai sisällönhallintaan liittyvien toimintojen ja näkymien toteuttaminen tyhjältä pöydältä sovelluskehykseen.

Siinä missä sisällönhallintajärjestelmät ja sovelluskehykset ovat jo vakiinnuttaneet paikkansa web-tuotannon parissa, sisällönhallintakehykset ovat edelleen suhteellisen tuoreita tulokkaita. Osittain juuri tästä syystä termin merkityksen määritteleminen tarkasti on hankalaa, etenkin kun monet sisällönhallintajärjestelmät luokittelevat itsensä nykyisin paitsi sisällönhallintajärjestelmiksi, myös sisällönhallintakehyksiksi.

Esimerkkejä sisällönhallintakehyksistä:

- **Drupal**, **Joomla**, **eZ Publish**, **MODx**, ja **WordPress** ovat sovelluskehyksen suuntaan laajennettuja sisällönhallintajärjestelmiä. Se, että järjestelmä on alkujaan suunniteltu sisällönhallintajärjestelmäksi, ei toki automaattisesti ole nega-

tiivinen asia, ja esimerkiksi Drupalin ja MODX:n kehittäjätyökalut ovatkin nykymittapuulla varsin monipuoliset.

- **ProcessWire**, **Symfony CMF**, ja **KeystoneJS** ovat sovellus- tai sisällönhallintakehyksiä sisällönhallintajärjestelmän ominaisuuksilla. Näistä kaksi jälkimmäistä pohjautuu sovelluskehyksiin: Symfony CMF PHP-pohjaiseen Symfony-sovelluskehykseen (SensioLabs 2015) ja KeystoneJS Node.js-pohjaiseen Express-sovelluskehykseen (KeystoneJS 2015).
- **Vegas CMF** edustaa viimeistä ja harvinaisinta ryhmää, eli sisällönhallintakehyksiä, jotka eivät sisällä lainkaan graafisia työkaluja. Vegas CMF:n pohjalla on kokonaisuudessaan PHP:n laajennuksena (C-kielellä) toteutettu Phalcon-sovelluskehys (Amsterdam Standard 2014).

2.5 Onnistunut järjestelmävalinta

Erityisesti monimutkaisempien toteutusten kohdalla oikein valittu alusta voi säästää merkittävästi työaikaa. Tästä näkökulmasta tarkasteltuna käytetty alusta kannattaisikin aina valita vain ja ainoastaan sen mukaan, mikä vaihtoehdoista kykenee parhaiten vastaamaan käsillä olevaan tarpeeseen, huomioiden niin järjestelmän tekniset edellytykset kuin projektin ja tilaajaorganisaation vaatimuksetkin.

2.5.1 Tapauskohtaisen valinnan haasteet

Toimittajat voivat kieltäytyä tarjoamasta muuta kuin omaa suosikkituotettaan, joka saattaa olla jokin yleisesti saatavilla olevista avoimista järjestelmistä tai vaihtoehtoisesti toimittajan itsensä kehittämä suljettu ratkaisu. Toisaalta myös tilaajat osaavat varsin usein vaatia tiettyjä tuotteita, joko hyvien perustelujen pohjalta, tai yksinkertaisesti siitä syystä, että tilaajalle tai heidän edustajalleen tuote on entuudestaan tuttu.

Toimittajalla on varmasti tutun tuotteen osalta parempi osaaminen, ja vastaavasti tilaajan sitoutumisen kannalta voi olla hyvinkin positiivinen vaikutus sillä, että he saavat vaikuttaa vahvasti tehtävään valintaan. Tietyn järjestelmän edellyttäminen projektin vaatimuksista riippumatta voi kuitenkin pahimmillaan johtaa tilanteeseen, jossa projekti joko epäonnistuu tai vähintäänkin monimutkaistuu tarpeettomasti.

Ikään kuin tässä ei olisi vielä tarpeeksi, myös ympäristö ja sopimustekniset seikat saattavat rajoittaa käytettävissä olevia vaihtoehtoja. Esimerkiksi tilaajaorganisaation noudattama kokonaisarkkitehtuurisuunnitelma saattaa edellyttää avoimen lähdekoodin tai tietyn tuotteen hyödyntämistä – tai muilla tavoin ohjata niitä valintoja, jotka liittyvät hankinnan kohteena olevaan tuotteeseen.

2.5.2 Tulevaisuuden ennustaminen on vaikeaa

Oman lisämausteensa järjestelmävalinnoille tuo se, että projektin ja tilaajan vaatimukset voivat muuttua, mistä johtuen tarpeiden arvioiminen etukäteen on haastavaa. Tehokkaan projektin- ja muutostenhallinnan lisäksi tähän voidaan kuitenkin reagoida esimerkiksi suosimalla sellaisia järjestelmiä ja alustoja, jotka ovat mahdollisimman joustavia, ja kykenevät tarvittaessa mukautumaan muuttuneisiin tarpeisiin.

Projektinhallinnassa puhutaan nykyisin paljon ketteristä menetelmistä, ja samaa ajatusta voidaan soveltaa verkkopalveluiden toteutukseen. Alkuperäinen projekti voidaan tässä ajatusmallissa mieltää pidemmän tähtäimen suunnitelmien ensimmäiseksi vaiheeksi, jossa toteutetaan yksinkertaisin käyttökelpoinen lopputuote (MVP, Minimum Viable Product). Vasta tämän ensimmäisen version pohjalta kerätty kokemus ja palaute näyttävät, mihin kehityspanos jatkossa kannattaa suunnata.

Paitsi nykyisten, myös tulevien tarpeiden huomioiminen on osa onnistunutta järjestelmävalintaa. Toteutetun ratkaisun jatkokehityksen tulisi olla paitsi mahdollista, myös mahdollisimman vaivatonta. Mikäli valittua alustaa on kankeaa tai kallista muokata jälkikäteen, voi jatkokehitys pahimmillaan tarkoittaa aloittamista tyhjältä pöydältä – ja tällöin on turhaa enää puhuakaan kustannustehokkuudesta.

2.5.3 Suunnittelu, avoimuus, ja ketteryys

Järjestelmävalintoihin liittyy monia haasteita, ja vaikkei näiden vaikutuksia välttämättä ole mahdollista – saati edes mielekästä – täysin kumota, tulisi ne kuitenkin tiedostaa päätöksiä tehtäessä. Kuten projektinhallinnassa yleensäkin, myös tässä tapauksessa mahdollisimman monien lopputulokseen vaikuttavien tekijöiden tiedostaminen on jo yksi askel kohti onnistunutta lopputulosta.

Huolellinen suunnittelu ja avoin dialogi tilaajan ja toimittajan välillä on toinen avain järjestelmävalinnan onnistumiseen: kaikki näkökulmat tulisi tuoda esiin varhaisessa vaiheessa projektia, jolloin päätöksiin voidaan vaikuttaa ilman kohtuuttomia lisäkustannuksia. Koska järjestelmävalinnalla on tärkeä rooli niin projektin aikana kuin sen jälkeenkin, kaikkien osapuolten olisi syytä olla valinnan kohteeseen tyytyväisiä.

Usein järjestelmävalintoja näkee perusteltavan markkinaosuuksilla. Tärkeämpää on kuitenkin tiedostaa, miksi tietty tuote on saavuttanut markkinoilla merkittävän roolin, ja ovatko tähän johtaneet tai tästä seuraavat tekijät käsillä olevan projektin kannalta hyödyllisiä. Suosio ei ole soveltuvuuden tae, ja voi olla jopa negatiivinen piirre: suosittu järjestelmä on yleensä suosittu myös hyökkääjien keskuudessa.

2.6 Yhteenveto ja pohdinta

Kehittäjien saatavilla oleva työkaluvalikoima on laajentunut merkittävästi kuluneiden vuosien ja vuosikymmenten aikana, ja vaikka monipuolinen valikoima onkin hyvä asia, aiheuttaa se myös väistämättä uusia haasteita.

Koska jokaista järjestelmää ei ole mahdollista hallita täydellisesti, kehittäjiltä kaivataan erikoistumista. Samanaikaisesti kuitenkin projektien vaihtelevat ja kasvavat tarpeet tekevät kuitenkin yhteen järjestelmään nojaamisen ongelmalliseksi. Yhdessä projektissa hyväksi koettu ratkaisu ei välttämättä sovellu toiseen alkuunkaan.

Sisällönhallintakehykset pyrkivät vastaamaan tähän haasteeseen tarjoamalla niin tehokkaat kehitystyökalut, käyttäjäystävälliset sisällönhallintatyökalut, kuin mukautuvat tietotyypit ja -rakenteetkin. Yhdistelemällä parhaita puolia eri kategorioista ne kykenevät vastaamaan erilaisiin tarpeisiin kustannustehokkaasti ja laadusta tinkimättä.

Projektiin osallistuvan asiantuntijan vastuulla puolestaan on varmistaa, että tarjottu ratkaisu palvelee projektin ja asiakkaan tarpeita. Tietyn järjestelmän suosittelu tilanteessa, jossa se ei selvästikään sovellu toteutettavan ratkaisun alustaksi – tai jossa on nähtävillä, että toinen järjestelmä olisi parempi valinta – on sekä epäeettistä että epäammattimaista.

3 PROCESSWIRE: ARKKITEHTUURI JA YDINKÄSITTEET

Oppaan ensimmäinen luku tutustuttaa lukija järjestelmän toimintaan ja hyödyntämiseen liittyviin ydinkäsitteisiin, sekä esittelee sen toteuttaman arkkitehtuurin lyhyesti. Toimintaympäristön tuntemus on tärkeä askel minkä tahansa uuden järjestelmän, tekniikan, tai taidon omaksumisprosessissa – ja kun perusteet on omaksuttu, on helpompaa soveltaa hankittua tietotaitoa niihin tilanteisiin, joihin ei löydy suoraa, valmista vastausta.

3.1 Luvun sisältö ja rakenne lyhyesti

Selkeyden vuoksi oppaan ensimmäisessä luvussa käsiteltävät asiat on jaettu neljään osaan: järjestelmän yleiset suunnitteluperiaatteet, sovellusarkkitehtuuri, sisältöarkkitehtuuri, ja kehittäjärajapinta. Tässä yhteydessä nämä esitellään pääasiassa teorian kautta, kun taas oppaan myöhemmät luvut pureutuvat esimerkiksi sisältöarkkitehtuuriin konkreettisten käytännön esimerkkien avulla.

3.2 Yhteenveto ja pohdinta

ProcessWire on yhdistelmä sisällönhallintakehystä ja sisällönhallintajärjestelmää. Sen valttikortteja ovat näkymien ja tietorakenteiden muokattavuus sekä kehittäjärajapinnan ja ylläpitotyökalujen käyttäjälähtöisyys. Kilpailevista tuotteista sen erottaa ennen kaikkea kyky suoriutua niin pienten sivustojen ja sovellusten kuin laajojen ja tietointensiivisten palveluidenkin tarpeista yksinkertaisuudesta ja helppokäyttöisyydestä tinkimättä.

Mikään ProcessWiren piirteistä ei ole sinänsä ennenkuulumaton, ja järjestelmän suunnittelussa onkin hyödynnetty surutta muista järjestelmistä lainattuja ideoita. Ainutlaatuiseksi ProcessWiren tekee kuitenkin se tapa, jolla kaikki tämä nivoutuu yhteen: yksikään järjestelmän piirteistä ei tunnu päälle liimatulta, vaan jokainen mukaan valittu palanen palvelee toinen toistaan ja sopii aukottomasti järjestelmän arkkitehtuuriin.

Tämän luvun asioista osaa käsitellään myöhemmissä luvuissa tarkemmin, mutta aivan kaikkeen ei kuitenkaan ole mahdollista pureutua kovin syvällisesti. Tämän kokoisessa järjestelmässä riittää jo melkoisesti asiaa käsiteltäväksi, ja esimerkiksi järjestelmän virallinen sivusto on kaikille ProcessWiresta kiinnostuneille erinomainen lisätietolähde.

4 SIVUPOHJAISTEN TIETORAKENTEIDEN PERUSTEET

Koska ProcessWire on sisällönhallintakehys, yksi sen tärkeimmistä tehtävistä on sisältötyyppien ja tietorakenteiden määrittelyn ja hallinnoinnin mahdollistaminen. Oppaan toisen luvun ensisijaisena tavoitteena on tarjota lukijalle perustason ymmärrys siitä, mitä sivut, sivupohjat, ja sivupuu oikeastaan ovat – ja miten ne liittyvät toisiinsa.

4.1 Luvun sisältö ja rakenne lyhyesti

Ensimmäisen luvun tapaan myös oppaan toinen luku alkaa teoriaosuudella, jonka tavoitteena on perehdyttää lukija hieman ProcessWiren tapaan käsitellä tietorakenteita sekä yksittäisiä tietueita. Teoriaosuuden jälkeen siirrytään soveltamaan teoriaa käytäntöön kuvitteellisen uutissivuston tietorakenteiden suunnittelua apuna käyttäen.

4.2 Yhteenveto ja pohdinta

ProcessWire on pohjimmiltaan sisällönhallintakehys, ja tietorakenteiden määrittely onkin yksi sen vahvuuksista. Uusille käyttäjille outo terminologia sekä omintakeinen tapa jäsentää asioita saattaa kuitenkin aiheuttaa päänvaivaa, ja ensimmäinen haaste onkin usein sen tosiseikan hahmottaminen, että sivu voi edustaa mitä tahansa kuviteltavissa olevaa tieto-oliota, ja käytetty sivupohja puolestaan määrittää kunkin sivun tietotyypin.

Sivupohjat koostuvat kentistä, joista kukin voi liittyä moneen eri sivupohjaan. Vaikka sivupohjien ja kenttien toteutustapa voikin alkuun vaikuttaa rajoittuneelta, pienen totuttelun jälkeen se käy järkeen. Järjestelmä ei varsinaisesti suosi kertakäyttöisiä sivupohjia tai kenttiä, mutta monikäyttöisyydestä on myös suoraa hyötyä: erinomaisen skaalautuvuuden lisäksi tietorakenteisiin voidaan koska tahansa tehdä muutoksia keskitetysti.

Sen lisäksi, että kaikki tietotyypit ovat vapaasti määriteltävissä, ProcessWire tarjoaa työkalut myös sivujen välisten relaatioiden hallinnointiin. Sivuliitokset mahdollistavat minkä tahansa sivun liittymisen mihin tahansa muuhun sivuun aina kun tälle on tarvetta. Kokonaisuudessaan järjestelmää voikin täysin rehellisesti suositella alustaksi niin monipuolisille ja tietointensiivisille sovelluksille kuin kaikenkokoisille sivustoillekin.

5 SIVUAIHIOIDEN TOTEUTTAMISEN PERUSTEET

Oppaan kolmannessa luvussa tavoitteena on lukijan perehdyttäminen sivuaihioiden toteuttamiseen ja sitä kautta tutustuttaminen ProcessWirelle ominaiseen tapaan erottaa sisältö ja esitystapa toisistaan. Siinä missä edellinen luku perehdytti lukijan tietorakenteiden käsittelyyn, keskitytään tässä luvussa puhtaasti sivuaihioiden tekniseen toteuttamiseen ja sen mukanaan tuomiin mahdollisuuksiin.

5.1 Luvun sisältö ja rakenne lyhyesti

Tässä luvussa teorian osuus on suppea ja lukija ohjataan varsin nopeasti käytännön esimerkkien pariin. Luvun alkupuoli keskittyy sivuaihioiden sijaintiin ja sisältöön yleisellä tasolla, minkä jälkeen esitellään käytännön esimerkkien kautta ProcessWiren kehittäjärajapinnan (API) toimintaa sekä sen erityispiirteitä, sekä esimerkiksi kuvien käsittelyä.

5.2 Yhteenveto ja pohdinta

ProcessWiren oliopohjainen, ketjutusta tukeva kehittäjärajapinta tekee sivun – ja oikeastaan koko sivuston – sisällön parissa toimimisesta vaivattoman ja miellyttävän kokemuksen. Rajapinnan oppimiskynnys on varsin alhainen ja luonnollisesti kynnys madaltuu entisestään, mikäli käyttäjällä on valmiiksi kokemusta esimerkiksi PHP-kielen ja muiden sisällönhallintakehysten tai -järjestelmien parissa työskentelystä.

ProcessWiren merkkausriippumattomuus tekee aihiotiedostojen laatimisesta erityisen joustavaa. Kukin sivuaihio voi HTML:n sijaan esittää sivun sisällön esimerkiksi XML- tai JSON-, YAML- tai PDF-muodossa. PHP:n keinoin kehittäjä voi lähettää haluamansa header-tietueet suoraan selaimelle, ja siten esimerkiksi kertoa sille, mitä tiedostomuotoa sivun sisältö oikeastaan edustaa (Tatroe, MacIntyre, ja Lerdorf 2013: 182.).

Sivuaihioiden laatiminen on niitä aiheita, joista oppaassa voisi kirjoittaa huomattavasti laajemminkin. Tämän luvun tarkoituksena on kuitenkin ensisijaisesti tärkeimpien kohtien nopea läpikäynti esimerkkien kautta, ja tältä osin tavoite täyttyy. Myöhemmissä luvuissa käsitellään erikseen sivuaihioiden hyödyntämistä hieman monimutkaisemmassa käyttötapauksessa esimerkiksi eräänlaisen MVC-mallin löyhän variaation kautta.

6 YKSINKERTAINEN MODEL-VIEW-CONTROLLER -MALLI

ProcessWire on erittäin joustava järjestelmä, joka mukautuu monenlaiseen tarpeeseen. Joustavuuden seurauksena kehittäjällä on paitsi mahdollisuus tehdä asiat haluamallaan tavalla, myös vastuu lopputuloksen toimivuudesta. Aihiotason rakennemalli on yksi aivan ensimmäisistä suunnitteluvaiheen päätöksistä, mutta vaikuttaa suoraan paitsi projektin onnistumiseen, myös toteutetun ratkaisun jatkokehitysmahdollisuuksiin.

6.1 Luvun sisältö ja rakenne lyhyesti

Myös tämä luku ohjaa lukijan lyhyen esipuheen jälkeen suoraan käytännön esimerkkien pariin. Lukijalle esitellään tämän luvun aikana kolme erilaista rakennemallia sivuaihioiden toteuttamiseen: Direct Output, Delayed Output, ja Model-View-Controller. Lisäksi luvussa kerrotaan jonkin verran kullekin rakennemallille ominaisista käyttötapauksista.

6.2 Yhteenveto ja pohdinta

Tässä luvussa esitelty rakennemalli on toki täysin käyttökelpoinen sellaisenaankin, mutta tämän oppaan perimmäinen tarkoitus ei kuitenkaan ole valmiiden ratkaisujen tarjoaminen, vaan mahdollisimman laajasti sovellettavissa olevan pohjatiedon tarjoaminen. Tässä kohtaa tärkeintä onkin ymmärtää, mihin kaikkeen ProcessWire alustana kykenee, ja miksei se pakota hyödyntämään esimerkiksi tiettyä valmista rakennemallia.

ProcessWire tekee lähes minkä tahansa ratkaisun toteuttamisen mahdolliseksi, mutta tukee samanaikaisesti myös kehittäjän työtä tarjoamalla pitkälle kehitettyjä toimintoja esimerkiksi sisällön hallintaan ja käyttäjien tunnistamiseen. Tietoisena päätöksenä järjestelmä ei kuitenkaan ota kantaa siihen, millä tavoin kehittäjä sitä päättää hyödyntää.

ProcessWiren päälle on toteutettu huomattavasti monipuolisempiakin variaatioita tässä luvussa esittelemästämme yksinkertaisesta MVC-mallista. Osa näistä on asennettavissa moduuleina ja osa toimitetaan sivustoprofiileina, mutta jos ProcessWire ja MVC yhdistelmänä vaikuttaa kiinnostavalta, kannattaa ehdottomasti tutustua näihin valmiisiin vaihtoehtoihin. Oman ratkaisun toteuttaminen alusta loppuun on erinomainen oppimiskokemus, mutta parempi lopputulos saavutetaan usein hyödyntämällä valmiita ratkaisuja.

7 VÄLIMUISTIRATKAISUJEN HYÖDYNTÄMINEN

Välimuisti eli ”cache” on tapa tallentaa valmiiksi noudettua ja käsiteltyä tietoa sellaiseen muotoon, josta se on myöhemmin otettavissa käyttöön aiempaa helpommin, nopeammin ja tehokkaammin (Wessels 2001). Välimuistiratkaisujen hyödyntäminen vaikuttaa erittäin olennaisesti minkä tahansa sivuston tai sovelluksen suorituskykyyn ja oppaan viidennessä luvussa keskitytään ProcessWiren tarjoamiin välimuistiratkaisuihin.

7.1 Luvun sisältö ja rakenne lyhyesti

Luvun alussa käsitellään välimuistiratkaisuja yleisellä tasolla sekä esitellään erään välimuistiratkaisun tuomia hyötyjä yksinkertaisen rasitustestin avulla hankittujen tulosten kautta. Suurin osa luvusta keskittyy kuitenkin esittelemään erityisesti ProcessWiren tarjoamia valmiita välimuistiratkaisuja, joiden kirjo onkin varsin monipuolinen.

7.2 Yhteenveto ja pohdinta

Välimuistiratkaisujen tehokas hyödyntäminen auttaa palvelemaan suurempia käyttäjämääriä nopeammin ja pienemmillä resursseilla. Suorien kustannussäästöjen lisäksi tällä on merkitystä niin palvelun käytettävyyteen kuin jopa sen näkyvyyteenkin, sillä esimerkiksi hakukoneet suosivat pääsääntöisesti nopeampia sivustoja. Samalla myös todennäköisyys palvelun estymiseen yllättävän ylikuormitustilanteen sattuessa pienenee.

Tässä luvussa esitellyt luvut puhuvat selkeää kieltä välimuistin hyödyntämisen tuomista eduista ja osoittavat, että parhaimmillaan kunkin sivupyynnön palvelemiseen voidaan käyttää jopa useita kymmeniä tai satoja kertoja vähemmän resursseja. Palvelun mittaluokan kasvaessa myös välimuistin käytöstä saadaan suuremmat hyödyt irti, mutta mitattavissa olevaa hyötyä sillä saavutetaan käytännössä lähes poikkeuksetta.

Koska välimuistiratkaisuja on useaan lähtöön, kehittäjän on hyvä olla selvillä kunkin hyödyistä voidakseen tehdä valistuneita päätöksiä asian suhteen. ProcessWire tarjoaa työkalut niin yksittäisten merkkijonojen kuin kokonaisten sivustojenkin käyttöön, ja kun mukaan lasketaan esimerkiksi ProCachen kaltaiset kolmannen osapuolen ratkaisut, jotakuinkin jokaiseen kuviteltavissa olevaan tilanteeseen on olemassa oma ratkaisunsa.

8 MODUULIKEHITYKSEN PERUSTEET

Modulaarisena järjestelmänä ProcessWire paitsi koostuu erillisistä moduuleista, on myös monipuolisesti laajennettavissa kehittäjän itsensä toimesta. Oppaan kuudes ja viimeinen luku keskittyy sivustokohtaisten moduulien kehittämiseen, sekä opastaa lukijan järjestelmän modulaarisen rakennemallin saloihin.

8.1 Luvun sisältö ja rakenne lyhyesti

Tässä luvussa käydään läpi uuden moduulin kehittämiseen liittyvät työvaiheet kohta kohdalta ja lopulta toteutetaan suhteellisen yksinkertainen – joskin sellaisenaan täysin käyttökelpoinen – esimerkkimoduuli. Heti luvun alkupuolella pureudutaan kuitenkin hieman tarkemmin siihen, mitä moduulit oikeastaan ovat, ja mihin niitä tarvitaan.

8.2 Yhteenveto ja pohdinta

Sivuaihioiden ja hallintanäkymien avulla on mahdollista toteuttaa erittäin monipuolisia ja laajoja ratkaisuja ilman pienintäkään tarvetta omille moduuleille. Moduulit kuitenkin mahdollistavat tiettyjä asioita, joita näillä tavoilla joko ei ole mahdollista toteuttaa, jotka olisivat kohtuuttoman työläitä. Kenties parhaimmillaan moduulit ovatkin juuri silloin, kun halutaan automatisoida ja yksinkertaistaa toistuvia, rutiininomaisia toimenpiteitä.

ProcessWiren moduulikehitys on tietoisesti suunniteltu välimuoto sovelluskehityksen parhaiden käytäntöjen ja helpon ymmärrettävyyden väliltä. Yksinkertaisiin tarkoituksiin suunnattujen moduulien laatiminen on todella helppoa ja onnistuu jopa ilman minkäänlaista aiempaa ohjelmointitaustaa – ja toisaalta vaikka aloittaminen onkin helppoa ja pienellä vaivalla saakin paljon aikaan, mahdollistaa järjestelmä myös laajojen moduulien toteuttamisen ja esimerkiksi moduulien välisten riippuvuussuhteiden määrittelyn.

Aloittelevan kehittäjän kannattaa tutustua valmiisiin moduuleihin, joita ProcessWiren moduulikirjastosta löytyy jo satoja. Luku ei toki ole erityisen vaikuttava esimerkiksi WordPressin kymmenien tuhansien lisäosien rinnalla, mutta kuitenkin niin laaja, että hyviä esimerkkejä löytyy taatusti. ProcessWiren moduulikirjastoon lisätyt moduulit on lisäysvaiheessa kevyesti auditoitu, joten laatukin on keskimäärin varsin hyvällä tasolla.

9 POHDINTA

Opinnäytteen laatimisen – ja erityisesti liitteenä olevan laajemman oppaan kirjoittamisen – yhteydessä suurimmiksi haasteiksi nousivat ehdottomasti aiheen rajaaminen sekä alkuperäisen, englanninkielisen terminologian kääntäminen suomenkieliseen muotoon ymmärrettävyyden kärsimättä.

Aiheen rajaamista puolestaan vaikeutti paitsi se, että lukijan tuntemusta etenkin aihepiirin teoreettisemmasta puolesta on vaikeaa ennakoida, myös se, että kohdejärjestelmä kehittyi myös kirjoitusprosessin aikana jatkuvasti. Kehittyvän järjestelmän dokumentointi on vaativa prosessi, joka vaatii tiettyjä rajanvetoja. Tällaisessa tapauksessa kirjallinen työ käsittelee käytännössä tiettyä vaihetta järjestelmän elinkaaressa.

Termien kääntäminen on puolestaan haasteellinen prosessi jo siitä yksinkertaisesta syystä, että kahdesta kielestä ei aina löydy täysin vastaavaa käsitteistöä, ja lähin yksiselitteinen käännös voi esimerkiksi yhden sanan sijaan olla kokonainen lause. Tässä tapauksessa kääntäminen vaikutti kuitenkin loogisimmalta, ja käännösten tavoitteena on myös tuoda kirjalliselle tuotokselle tiettyä uskottavuutta ja luontevuutta.

Kokoelma teknisiä termejä sekä moduulien ja työkalujen nimiä sen sijaan jäi kääntämättä. Syynä tähän oli joko se, että järjestelmässä nämä esiintyvät aina alkuperäisessä muodossaan, tai vaihtoehtoisesti se, että kyse on oikeastaan erisnimestä. Vaikka suomalaisittain voikin kuulostaa hieman oudolta, kun tekstin seassa mainitaan WireShell-työkalu, MarkupCache-moduuli tai Module-rajapinta, tälle on olemassa varsin hyvät perusteet.

Mitä työn taustalla olevaan järjestelmään tulee, työn laatiminen pakotti tutustumaan juurta jaksain niin järjestelmän suunnitteluperiaatteisin, arkkitehtuuriin, kuin sen ympärille rakentuneeseen ekosysteemiinkin. Mikään mainituista ei ollut varsinaisesti vierasta ennen työn aloittamistakaan, mutta kirjoittaminen pakotti tutustumaan myös sellaisiin aihepiireihin, joihin perehtyminen on aiemmin jäänyt vähemmälle.

Alkujaan opinnäytteen yhteyteen oli tarkoitus mahduttaa esimerkiksi haastattelujen ja kyselyiden perusteella hankittua lisänäkemystä, mutta lopulta käytännön tarpeet ohjasivat työn lähemmäs perinteistä käsikirjamaista esitystapaa. Samasta syystä työn tutkimuksellisen perustan muodostaa empiirinen tutkimus ja kvalitatiivinen lähestymistapa.

Toisena konkreettisena esimerkkinä kysely- ja haastattelututkimusten lisäksi työssä olisi ollut mahdollista hyödyntää vielä laajemmin benchmarking-tyyppistä lähestymistapaa, eli vertailuanalyysiä. Tästä jouduttiin luopumaan paitsi aikarajoitteiden johdosta, myös siitä syystä, että tällöin työn pituus olisi todennäköisesti kasvanut entisestään, mikä taas ei ollut tässä yhteydessä enää toivottavaa.

Vaikkei tämän opinnäytteen lopputuloksena syntynytkään erityisen mullistavaa uutta tietoa tai täysin aiemmasta poikkeavaa uutta näkemystä, tässä kohtaa on syytä huomioida, että tämä ei ollut missään vaiheessa edes tarkoituksena. Työn tavoite ja tarkoitus oli luoda hyödyllistä ja erityisesti uusien käyttäjien tarpeet huomioiva perehdyttämismateriaali, ja tämä tavoite täyttyi varsin hyvällä tasolla.

Käytäntö tulee lopulta näyttämään, missä määrin työn lopputulosta tullaan lopulta hyödyntämään. Hyvän referenssimateriaalin merkitys niin työn tehostajana, kustannusten alentajana, kuin motivaation parantajanakin on kuitenkin kiistaton, ja toivottavaa on, että myös tämän opinnäytteen lopputuloksena syntyneellä materiaalilla on edellä mainittuja vaikutuksia työn tilaajaorganisaation sisällä.

Vaikka opinnäytteen tärkeimmät tavoitteet täyttyivätkin, tässä yhteydessä alkuun saatettua opasta on ehdottomasti tarkoitus jatkossa laajentaa ja kehittää. Ne aiheet, jotka tässä vaiheessa jouduttiin alkujaan karsimaan opinnäytteelle mielekkään pituuden saavuttamiseksi, tullaan lisäämään tähän tekstiin, ja kokonaisuus on myöhemmin tarkoitus julkaista sähköisenä opaskirjana avoimen lisenssin alla.

LÄHTEET

Amsterdam Standard 2014. Vegas Content Management Framework. Luettu 7.10.2015. http://vegas-cmf.github.io/

Cake Software Foundation 2015. CakePHP Cookbook Documentation. Luettu 7.10.2015. http://book.cakephp.org/3.0/_downloads/en/CakePHPCookbook.pdf

KeystoneJS 2015. Luettu 7.10.2015. http://keystonejs.com/

Mitchell, R. 2013. Choosing an open-source CMS, part 1: Why we use Drupal. Luettu 7.10.2015. http://www.computerworld.com/article/2494786/e-commerce/choosing-an-open-source-cms--part-1--why-we-use-drupal.html

Phalcon Team 2015. Phalcon PHP Framework Documentation. Luettu 7.10.2015. https://media.readthedocs.org/pdf/phalcon-php-framework-documentation/latest/phalcon-php-framework-documentation.pdf

Severdia, R. 2015. Joomla! - Core Features. Luettu 7.10.2015. https://www.joomla.org/core-features.html

Tatroe, K., MacIntyre, P. ja Lerdorf, R. 2013. Programming PHP. O'Reilly Media, Inc.

Wessels, D. 2001. Web Caching. O'Reilly Media, Inc.

WordPress.org 2015. Features. Luettu 7.10.2015. https://wordpress.org/about/features/

Zend Technologies 2015. About - Zend Framework. Luettu 7.10.2015. http://framework.zend.com/about/

LIITTEET

Liite 1. ProcessWire-järjestelmän perusteet kehittäjille.

ProcessWire-järjestelmän perusteet kehittäjille

Koivula Teppo

Versio 1.0

11.12.2015

SISÄLLYS

1 PROCESSWIRE: ARKKITEHTUURI JA YDINKÄSITTEET

Tässä luvussa käymme tiiviisti läpi tärkeimmät ProcessWiren toimintaan ja hyödyntämiseen liittyvät käsitteet. Sivustojen rakentaminen ei välttämättä edellytä kovin laajaa ymmärrystä siitä, miten järjestelmä todellisuudessa toimii, mutta toiminnan ymmärtäminen syvemmällä tasolla on erittäin hyödyllistä esimerkiksi ongelmien selvittelyn ja vikojen etsimisen näkökulmasta.

1.1 Yleiset suunnitteluperiaatteet

Suunnitteluperiaatteet ovat päätöksiä, arvoja, ja mielikuvia, jotka paitsi ohjaavat järjestelmän kehitystä, myös erottavat sen kilpailijoistaan. Suunnitteluperiaatteiden rinnalla esimerkiksi tekninen toteutus on toissijainen ominaisuus: järjestelmän taustalta löytyvä teknologia muuttuu ja kehittyy, suunnitteluperiaatteet pysyvät ennallaan.

1.1.1 Joustavuus ja räätälöitävyys

ProcessWiren ensimmäinen ja tärkein suunnitteluperiaate on joustavuus. Sen lisäksi, että tietotyypit, kentät, sivurakenteet jne. ovat sivusto- ja sivukohtaisesti määriteltävissä, järjestelmä ei sido myöskään kehittäjää tiettyyn toimintamalliin tai rakenteeseen, vaan mahdollistaa erilaisten toteutustapojen hyödyntämisen:

- Aihiotasolla on oletuksena käytössä "yksi näkymätiedosto sivupohjaa kohti" -ratkaisu, mutta pienellä vaivalla tämän tilalle voidaan vaihtaa myös jokin muu sovelluskehityksen rakennemalli; esimerkiksi MVC (Model-View-Controller), MTV (Model-Template-View), tai MVVT (Model-View-ViewTemplate).
- Koukkujen (hooks) avulla voidaan ohjelmallisesti vaikuttaa järjestelmän toimintaan, muokata metodien argumentteja ja paluuarvoja, korvata haluttu metodi kokonaisuudessaan, ja jopa lisätä järjestelmään kokonaan uusia toiminnallisuuksia.
- Modulaarinen rakenne mahdollistaa paitsi uusien toimintojen, myös esimerkiksi uusien hallintanäkymien ja tietorakenteiden käyttöönoton vaivattomasti asennettavien moduulien avulla.

1.1.2 Rajapinta kaiken keskiössä

ProcessWiren juuret ovat sisällönhallintakehyksessä, jonka päälle on toteutettu sisällönhallintajärjestelmä. Useimmat sivustot hyödyntävät molempia piirteitä samanaikaisesti ja näiden erottaminen toisistaan olisikin jo käytännössä hyvin vaikeaa, mutta tällä on kuitenkin ollut merkittävä vaikutus järjestelmän rajapinnan suunnittelussa.

Kaikki mikä on mahdollista käyttöliittymien kautta, on tehtävissä myös ohjelmallisesti kehittäjärajapinnan ja valitsinmoottorin avulla. Kehittäjärajapinta paitsi helpottaa kehittäjän työtä, myös parantaa järjestelmän turvallisuutta, sillä esimerkiksi raakoja SQL-kyselyitä tarvitaan tässä arkkitehtuurissa äärimmäisen harvoin.

1.1.3 Merkkausriippumattomuus

ProcessWire ei ota kantaa siihen, millaista sisältöä sen avulla loppukäyttäjille esitetään. Sivustoa toteuttava kehittäjä laatii yleensä suurimman osan HTML-merkkauksesta itse, järjestelmä puolestaan mahdollistaa sivun kenttiin tallennettujen tietueiden tulostamisen merkkauksen sekaan haluttuihin kohtiin.

ProcessWiren tuottama sisältö ei rajoitu myöskään HTML-merkkaukseen, vaan se voi tarjoilla sisällön esimerkiksi JSON- tai XML-muodossa. Tämä on käytännöllistä, mikäli sivustoa halutaankin perinteisen web-sivuston sijaan hyödyntää esimerkiksi REST-rajapinnalla varustettuna tietovarantona, tai vaikkapa RSS-syötekokoelmana.

Merkkausriippumattomuus ei sulje pois erikseen kutsuttavia, merkkausta tuottavia moduuleja. Näille on jopa oma moduuliluokkansa: Markup-moduulit. Tyypillinen Markup-moduuli ottaa parametrina valmiin tietorakenteen tai muuttujan, ja luo sen pohjalta esimerkiksi valikkorakenteen, RSS-syötteen, tai vaikkapa PDF-tiedoston.

1.1.4 Ei omaa aihiointikieltä

ProcessWiren tapauksessa kaikki sovellus- ja aihiologiikka ydinkoodista sivuaihioihin kirjoitetaan PHP:llä, joka on suosittu ja suhteellisen yksinkertainen ohjelmointikieli. ProcessWire ei siis sisällä omaa aihiointikieltään, eikä myöskään oletuksena tue ainuttakaan erityisesti aihiointikäyttöön suunnitelluista valmiista kielistä (Twig, Smarty, jne.)

Koska aihiointikielenä on PHP, kehittäjällä on toisaalta enemmän vastuuta tekemisistään, mutta samalla myös mahdollisuudet kaikkeen siihen, mihin täysiverinen ohjelmointikieli kykenee. PHP:n perusteet ovat paitsi helposti opittavissa, myös laajemmin hyödynnettävissä jatkoa ajatellen kuin puhtaasti aihiointitarkoituksiin erikoistuneen kielen.

1.1.5 Ei vain web-käyttöön

ProcessWire on mahdollista käynnistää ns. bootstrap-menetelmällä minkä tahansa muun PHP-sovelluksen sisään. Käytännössä tämä tarkoittaa bootstrap-tiedoston (index.php) lataamista esimerkiksi PHP:n include-lausekkeella, jolloin järjestelmä käynnistetään ja sen kehittäjärajapinta tuodaan bootstrap-tiedoston ladanneen ohjelman saataville.

Koska kehittäjärajapinnan yli on mahdollista tehdä lähes mitä tahansa, bootstrap-menetelmä mahdollistaa esimerkiksi järjestelmän hyödyntämisen toisen sovelluksen tietorakenteiden ja sisältöjen hallintaan, ja on tehnyt mahdolliseksi myös esimerkiksi WireShell-nimisen erillisen sovelluksen, joka on käytännössä helppokäyttöinen ProcessWire-komentokehote.

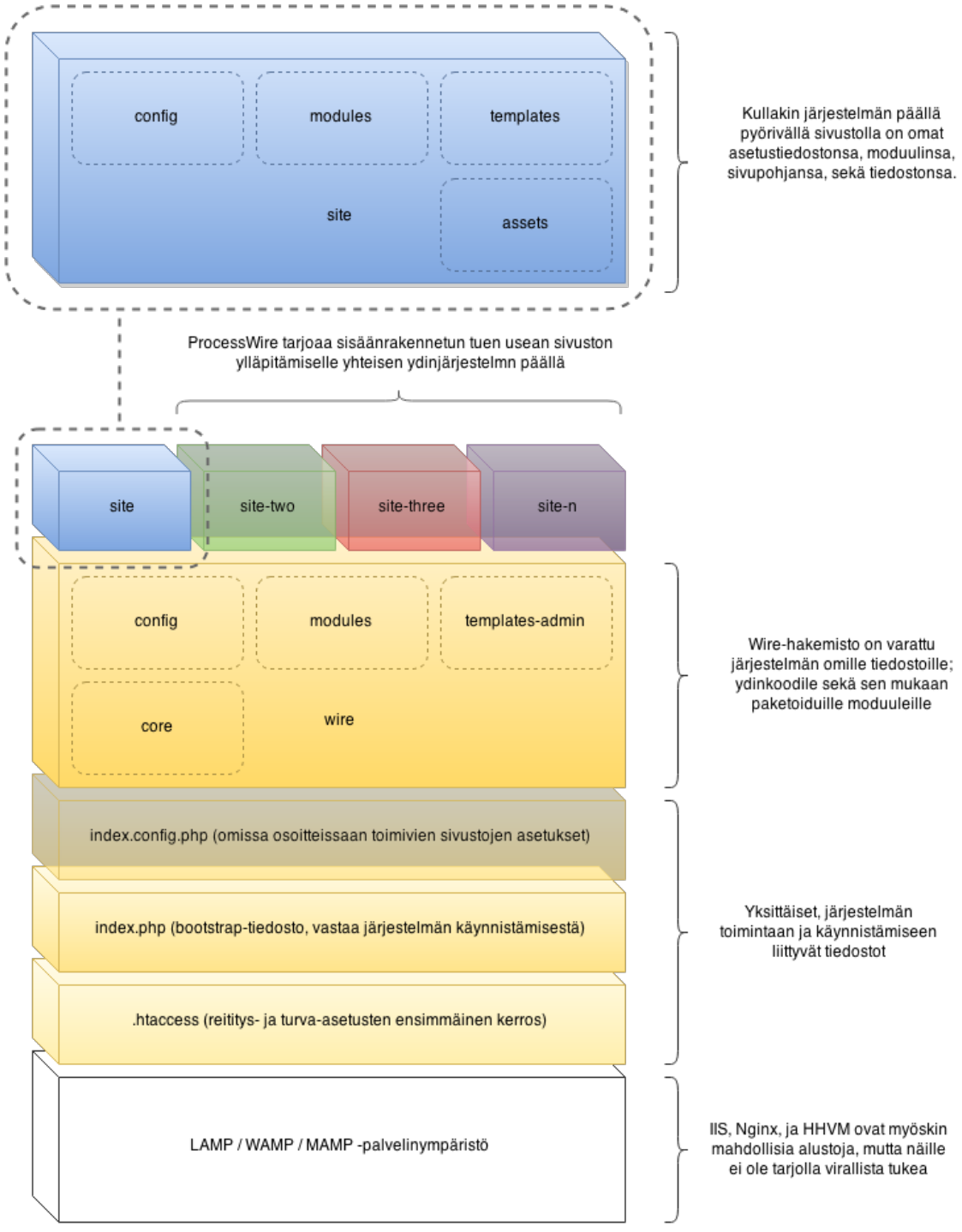
1.2 Sovellusarkkitehtuuri

ProcessWire perustuu oliopohjaiseen PHP-sovelluskoodiin ja edellyttää toimiakseen sekä tietokantaohjelmistoa (MySQL tai MariaDB), että web-palvelinohjelmistoa (yleensä Apache, mutta vaihtoehtoisesti myös IIS tai nginx). Tarkemmat ja ajantasaisemmat järjestelmävaatimukset löytyvät järjestelmän viralliselta sivustolta; esimerkiksi versionumerot vaihtelevat, eikä niiden kirjaaminen tähän yhteyteen siten ole mielekästä.

Järjestelmä on modulaarinen ja koostuu joukosta ydinkomponentteja (core) sekä niiden päälle asennettavista moduuleista (modules). Uusien moduulien toteuttamista varten ProcessWire tarjoaa omat moduulirajapintansa, minkä lisäksi valmiita, verkkopohjaisesta moduulikirjastosta vapaasti saatavilla olevia moduuleja voidaan ladata ja asentaa suoraan hallintakäyttöliittymän kautta.

Moduulit voivat esimerkiksi laajentaa järjestelmää uusilla ominaisuuksilla ja hallintanäkymillä, mutta myös vaikuttaa olemassa olevien ominaisuuksien toimintaan kouk-

kuja (hooks) hyödyntäen. Koukkujen hyödyntäminen on mahdollistettu myös muilla tavoin kuin moduulien avustuksella, mutta näihin perehdymme tarkemmin hieman myöhemmin.



KAAVIO 1. ProcessWire-järjestelmän arkkitehtuuri kerroksittain.

1.2.1 Järjestelmän käynnistäminen

ProcessWiren käyttöympäristö on verkkopalvelin, jossa palvelinohjelmisto (Apache) ohjaa selaimen pyynnöt järjestelmän käsiteltäviksi. Ensimmäinen vaihe kyselyiden käsittelyssä on tällöin .htaccess-tiedosto, joka estää suoran pääsyn järjestelmän hakemistoihin ja niistä löytyviin PHP-tiedostoihin, suodattaa virheelliset tai levyltä löytyviin staattisiin resursseihin kohdistuvat pyynnöt, sekä muotoilee kyselyä tarpeen mukaan.

Aikaisessa vaiheessa tehtävä suodatus on samalla ensimmäinen taso järjestelmän tietoturva-arkkitehtuurista, ja sen ansiosta epäilyttävistä kyselyistä merkittävä osa karsiutuu pois jo ennen kuin ne päästetään PHP:n käsiteltäviksi. Tällaisista kyselyistä voidaan joko poistaa tarpeettomat tai suoranaisesti vaaralliset merkit (Zalewski 2012: 40), tai niihin voidaan vastata suoraan 404-virheviestillä, hieman tilanteesta riippuen.

Tietoturvasyiden lisäksi .htaccess-tasolla tehtävät toimenpiteet ovat etenkin muistin käytön kannalta tehokkaita, koska PHP voidaan sivuuttaa kokonaan, mikäli kysely kohdistuu esimerkiksi levyllä sijaitsevaan staattiseen tiedostoon. PHP:n rooli alkaakin vasta siinä vaiheessa, kun kysely edellyttää esimerkiksi tietokantayhteyksien muodostamista.

Ensimmäisen käsittelykierroksen jälkeen kyselyt ohjataan bootstrap-tiedostolle (index.php), jonka tärkein tehtävä on järjestelmän käynnistäminen. Bootstrap-tiedoston lataaminen mahdollistaa itse asiassa ProcessWiren käynnistämisen myös web-kontekstin ulkopuolella, mistä syystä ProcessWire soveltuukin mainiosti kaikkiin palvelinpäässä tapahtuvaa automaatiota vaativiin tarpeisiin.

1.2.2 Hakemistorakenne

Hakemistotasolla ProcessWiren muodostuu kahdesta osasta, jotka tunnetaan nimillä wire ja site. Siinä missä wire pitää sisällään järjestelmän komponenteista koostuvan ydinkoodin (core) sekä sen mukaan paketoidut moduulit (core modules), site sisältää yksittäisen sivuston asetuksineen (site config), sivupohjineen (templates) ja moduuleineen (site modules).

```
.
├── site
|   ├── assets
|   |   ├── cache
|   |   ├── files
|   |   ├── logs
|   |   └── sessions
|   ├── modules
|   └── templates
└── wire
    ├── core
    ├── modules
    └── templates-admin
```

KAAVIO 2. ProcessWire-sivuston hakemistorakenne yleisellä tasolla.

Wire-hakemistoon ei ole suositeltavaa tehdä omia muutoksia. Syynä tähän on se, että järjestelmän päivittäminen tapahtuu tyypillisesti korvaamalla wire-hakemisto kokonaisuudessaan uudella versiolla. Myös tietyt sivuston juuresta löytyvät tiedostot saatetaan päivityksen yhteydessä korvata, mutta tämä on harvinaisempaa, ja esimerkiksi .htaccess-tiedostoon on usein paitsi suotavaa, myös tarpeellista tehdä omia muutoksia.

Siinä missä wire on syytä jättää rauhaan, site-hakemisto puolestaan toimii kehittäjän omana leikkikenttänä. Sen sisältä löytyvät sivustokohtaiset tiedostot asetuksista (/site/config.php) sivupohjiin (/site/templates/). Sivustokohtaisia site-hakemistoja voi yhdessä ProcessWire-asennuksessa itse asiassa olla useita, sillä järjestelmässä on sisäänrakennettu tuki useille sivustoille; tällöin jokainen niistä koskee yhtä sivustoa.

1.2.3 Tietoturva

Tietoturva on tärkeä osa minkä tahansa Web-sovelluksen sovellusarkkitehtuuria. ProcessWiren tapauksessa tietoturva on usein mainittu yhtenä järjestelmän tärkeimmistä piirteistä, ja jopa sen vahvuutena. ProcessWire huolehtii sen avulla toteutettujen ratkaisujen tietoturvasta muun muassa seuraavien menetelmien ja työvälineiden avulla:

- Sisäänrakennettu CSRF-suojaus. CSRF viittaa hyökkäykseen, jossa ulkopuolinen sivusto suorittaa käyttäjän selaimen välityksellä esimerkiksi ylläpitoon liittyviä toimintoja kohdesivustolla (Zalewski 2012: 84, 262; Shiflett 2006: 24-28).

- SQL-injektioiden riskiä vähentävä valitsinmoottori. SQL-injektiossa hyökkääjä huijaa sivuston suorittamaan vapaamuotoisia SQL-komentoja esimerkiksi huonosti sanitoitujen parametrien avulla (Zalewski 2012: 265; Shiflett 2006: 35-39).
- Epäilyttävien URL-osoitteiden siivoaminen automaattisesti .htaccess-tasolla jo ennen kuin ne ehtivät PHP:n käsiteltäviksi. Mikäli käyttäjän antamia parametreja ei siivota, tästä voi seurata esimerkiksi se, että hyökkääjä pääsee käsiksi suojattuihin tietoihin (Zalewski 2012: 40, 265; Shiflett 2006: 57-59).
- Järjestelmän omien tiedostojen ja hakemistojen suojaus ulkopuolisilta selaajilta .htaccess-sääntöjen avulla. Ilman tällaista suojausta hyökkääjä voisi päästä esimerkiksi lukemaan sivuston lähdekoodia, tai joissain tapauksissa periaatteessa jopa suorittamaan sivustolla ei-sallittuja komentoja (Shiflett 2006: 51-53).
- XSS-hyökkäysten riskin vähentämiseksi vain ylläpitäjät pääsevät kiinni käyttöliittymiin, joiden kautta voidaan tallentaa merkkausta. Järjestelmän mukana toimitettava HTMLPurifier-moduuli mahdollistaa lisäksi merkkauksen siistimisen. XSS-hyökkäyksessä luotetulle sivustolle ujutetaan omia skriptejä, jotka saattavat vierailijan vaaratilanteeseen (Zalewski 2012: 262; Shiflett 2006: 23).
- Salasanat tallennetaan tietokantaan salattuina. Salausmenetelmänä hyödynnetään vahvaa Blowfish-salausta yhdistettynä kahteen salt-tietueeseen (sivusto- ja käyttäjäkohtaiseen). Salasanojen tallentaminen salaamattomina on turvallisuusriski, joten niiden osalta tulisi aina hyödyntää vahvaa salausta (Shiflett 2006: 98-100).
- Kirjautumisen yhteydessä on oletuksena käytössä aikaviive, joka aktivoituu, kun tunnukselle syötetään väärä salasana. Näin vaikeutetaan merkittävästi salasanojen koneellista, arvailuun perustuvaa murtamista (Shiflett 2006: 64-67).
- Kirjautumisistunnot on oletuksena evästeen lisäksi sidottu käyttäjän selaintunnisteeseen sekä IP-osoitteeseen. Näin vaikeutetaan istuntojen kaappausta, sekä useita muita istuntojen käsittelyyn liittyviä riskejä (Shiflett 2006: 43-50).
- Istuntojen säilöminen tietokantaan on sisäänrakennettu toiminto. Esimerkiksi jaetussa ympäristössä istuntotietojen levylle PHP:n oletussijaintiin voi olla riskialtis toimenpide (Tatroe, MacIntyre, ja Lerdorf 2013: 199.)
- Välimuistiratkaisut eivät oletuksena tallenna välimuistiin tietoja, joihin käyttäjä on parametrien avustuksella voinut vaikuttaa. Ilman tällaista suojausta hyökkääjä saattaisi kyetä myrkyttämään sivuston välimuistin, eli ujuttamaan sivustolle haitallista tai harhaanjohtavaa sisältöä (Zalewski 2012: 263).

- Kaikki tietyn sivuston (mahdolliset) osoitteet tallennetaan config-tietueeseen httpHosts, ja näiden avulla voidaan varmistua esimerkiksi siitä, että sivustoa ei koskaan tarjoilla sellaisesta osoitteesta, josta sitä ei pitäisi löytyä.
- Sanitizer-luokan – ja siihen sidotun $sanitizer-API-muuttujan – tarkoituksena on tehdä esimerkiksi käyttäjän antaman syötteen (GET- ja POST-parametrien sekä URL-segmenttien) siistiminen ja sanitointi kehittäjän kannalta helpoksi.

Yllä mainitut tietoturvaominaisuudet ovat vain murto-osa ProcessWiren tietoturva-arkkitehtuurista. Tarkoituksena ei siis ole listata kaikkia järjestelmän tietoturvaan liittyviä piirteitä, vaan ainoastaan osoittaa, että ProcessWiren suunnittelussa tietoturvalla on aina ollut tärkeä rooli. ProcessWiren tapauksessa tietoturva ei ole jälkiajatus, vaan kiinteä ja kriittisen tärkeä osa järjestelmän sovellusarkkitehtuuria.

1.2.4 Monikielisyys

ProcessWire 2.2 sisälsi ensimmäistä kertaa tuen monikielisyydelle (Cramer 2012), ja sittemmin kielitukea on kehitetty ahkerasti. Nykyisin monikielisyys tarkoittaa kolmea ominaisuutta, joita voidaan hyödyntää yhdessä tai erikseen, sivuston tarpeiden mukaan:

- Koodin monikielisyyden (code internalization) avulla kehittäjä voi määrittää aihiotiedostoissa tai moduuleissa käännettäviä termejä, jotka tunnistetaan ja voidaan kääntää hallintakäyttöliittymästä löytyvän käännöstyökalun avulla.
- Monikielisillä kentillä voi olla jokaiselle kielelle oma arvonsa. Monikielisyyttä tukevien kenttien lisäksi mille tahansa kentälle voidaan määritellä kielikohtaisia versioita (language alternate fields) luomalla saman niminen kenttä ja lisäämällä nimen loppuun kielen nimi (esim. title_english, title_swedish, jne.)
- Monikieliset sivujen nimet (multi-language URLs) mahdollistavat sivujen esittämisen eri osoitteissa kielen perusteella – tai eri kielellä osoitteen mukaan, tulkinnasta riippuen: sivu /yritys/ voidaan esittää englanninkielisenä osoitteessa /en/company/ ja ruotsinkielisenä osoitteessa /se/foretaget/, jne.

Käytössä olevat kielet määritellään hallintakäyttöliittymän kautta. Ellei kieltä luoda suoraan valmiista kielipaketista, se on tyhjä, eli vailla ainuttakaan käännöstä. Kielten nimiä ei ole sidottu mihinkään standardiin tai virallisiin kieliin, ja niinpä esimerkiksi Suomi, English, Svenska, Java, ja Perl ovat kaikki täysin hyväksyttäviä kielten nimiä.

1.2.5 Monisivustotuki

ProcessWiren sisäänrakennetussa monisivustotuessa kullakin sivustolla on omat site-hakemistonsa, omat aihionsa, omat moduulinsa, sekä omat tietokantansa, mutta järjestelmän ydinkoodi on kuitenkin kaikille yhteinen. Käytännössä kyse on siis erillisistä sivustoasennuksista yhden ytimen päällä, josta merkittävimpänä etuna seuraa se, että ydinkoodin päivittäminen onnistuu kerralla ja keskitetysti.

Vaihtoehtoinen tapa monisivustoasennuksen toteuttamiseen on kolmannen osapuolen Multisite-moduuli, joka mahdollistaa yhteisen tietokannan ja site-hakemiston jakamisen usean sivuston kesken. Myös sivustojen hallinta on tällöin yhteinen, eli kaikkia sivustoja hallinnoidaan yhden sivupuun kautta. Tällöin yksi sivustoista on aina pääsivusto, muut sivustot puolestaan näyttäytyvät omina haaroinaan sivupuussa.

```
Etusivu
├── Sisältösivu 1
├── Sisältösivu 2
|
├── subdomain.example.com
|   ├── Sisältösivu 1
|   └── Sisältösivu 2
|
└── www.anotherdomain.com
    ├── Sisältösivu 1
    └── Sisältösivu 2
```

LISTAUS 1. Multisite-moduulin hyödyntämä sivurakenne.

Molemmissa malleissa kullakin sivustolla on oma verkko-osoitteensa, ja ulkopuolisen selaajan näkökulmasta kyse on kaikin puolin toisistaan irrallisista sivustoista. Se, kumpi malleista sopii kuhunkin tilanteeseen, määräytyy pitkälti sen pohjalta, mitä monisivustoasennuksella tarkalleen tavoitellaan:

- Multisite-moduulin etuna on sivustojen yhteinen hallinta ja mahdollisuus aihiotiedostojen, moduulien, jne. helppoon jakamiseen sivustojen kesken. Toisaalta tämä on myös Multisite-moduulin haaste, sillä esimerkiksi oikeuksien rajaami-

nen ja sivustokohtaisten sivupohjien, moduulien, ja aihiotiedostojen mahdollistaminen aiheuttaa tällöin ylimääräistä päänvaivaa.

- Sisäänrakennetun monisivustoasennuksen etuna on ydinkoodin keskitetty päivitys, mutta juuri muuta merkittävää etua tällä ei saavuteta. Toisaalta sivustojen kehittäminen erillään on tällöin yksinkertaisempaa, koska niiden välillä ei ole ydinkoodin lisäksi muita ylimääräisiä sidoksia.

Molemmat mallit tarjoavat tehokkaan tavan hallinnoida useampaa sivustoa keskitetysti, mutta toisaalta esimerkiksi ydinkoodin päivittäminen yhdelle sivustolle muiden jäädessä ennalleen on käytännössä mahdotonta. Monisivustomallin soveltuvuus tulisikin aina arvioida huolellisesti, sillä vaikka sivustojen erottaminen toisistaan onnistuu jälkikäteenkin, tästä seuraa lisätyötä ja usein myös katkoksia sivustojen toimintaan.

1.2.6 Moduulit ja moduuliluokat

ProcessWiren oletusasennukseen kuuluu ydinkoodin lisäksi kokoelma valmiiksi mukaan paketoituja moduuleja. Suurin osa mukaan paketoiduista moduuleista liittyy joko hallintanäkymiin, kenttien graafisin muokkausnäkymiin, tai kenttätyyppeihin.

Järjestelmään on saatavilla myös kolmannen osapuolen moduuleja, joiden tiedot on koottu julkiseen moduulihakemistoon osoitteessa http://modules.processwire.com/. ProcessWiren moduulikokoelma on merkittävästi suppeampi kuin esimerkiksi WordPress-järjestelmän vastaava: siinä missä WordPress-lisäosia oli tätä kirjoitettaessa kymmeniä tuhansia, ProcessWire-moduulien kohdalla puhutaan muutamista sadoista kappaleista.

Moduulien rajallinen lukumäärä on toisaalta mahdollistanut hakemistoon lisättyjen moduulien suhteellisen tarkan läpikäynnin, laadunvarmistuksen, ja hyväksymisprosessin. Moduulien luotettavuus on varsin tärkeä yksityiskohta, sillä esimerkiksi WordPressin osalta suurin osa viimeaikaisista tietoturvaongelmista on WP White Securityn laatiman selvityksen perusteella liittynyt juuri kolmannen osapuolen moduuleihin (Abela 2014).

ProcessWire-moduulit on jaettu luokkiin tarkoituksensa ja toimintaperiaatteensa perusteella. Kullakin moduuliluokalla on oma, ennalta määritelty tarkoituksensa järjestelmän elinkaaressa: Process-moduulit vastaavat ylläpitonäkymistä, Markup-moduulit tuottavat

merkkausta, WireMail-moduulit mahdollistavat sähköpostin, Fieldtype-moduulit määrittelevät kenttätyypit, ja niin edelleen.

1.3 Sisältöarkkitehtuuri

ProcessWiren tärkeimmät käsitteet sisältöarkkitehtuurin näkökulmasta ovat sivu (page), sivupohja (template), ja kenttä (field). Käsittelemme tietorakenteiden hyödyntämistä käytännössä omassa luvussaan, tässä luvussa perehdymme lähinnä ProcessWire-sivustojen sisältöarkkitehtuurin perusteisiin käsitetasolla:

- Sivu on yleiskäyttöinen sisältöolio. Jokainen sivu noudattaa tiettyä sivupohjaa, joka määrittää kyseisen sivun kentät, eli toimii sivun tietotyyppinä.
- Sivupohja on kokoelma kenttiä sekä muuta sivujen tarvitsemaa metadataa. Sivupohjien avulla hallitaan myös esimerkiksi sivuston oikeusrajauksia.
- Kenttä on tietyn kenttätyypin ilmentymä, ja mahdollistaa sekä tiedon tallentamisen halutussa muodossa tietokantaan, että noutamisen käyttöä varten.

1.3.1 Sivut (pages)

ProcessWiren sivu vastaa käsitteenä pitkälti Drupal-järjestelmän solmua, MODX:n resurssia (Reid 2014), sekä Joomlan artikkelia. Syy siihen, että termiksi on valittu juuri sivu, löytyy kontekstista: ProcessWire on ensisijaisesti suunniteltu web-käyttöön, ja web-ympäristössä sisältöä selataan juurikin sivuina.

Jokaisella sivulla, oli kyseessä sitten julkinen ja selaajalle näytettäväksi tarkoitettu sivu tai enemmänkin metadataa (esimerkiksi kategoria), on aina olemassa oma URL-osoitteensa, joka perustuu sivun sijaintiin sivupuussa sekä sivukohtaiseen name- eli nimi-tietueeseen. Tämä palvelee kahta tarkoitusta:

- Sivun URL-osoite on uniikki tietue, jonka perusteella sivu on yksilöitävissä. Sivun nimi sen sijaan ei ole uniikki, sillä muista sivuhaaroista voi löytyä saman nimen omaavia sivuja. Sivuilla on myös numeeriset, uniikit ID-arvonsa, mutta selkokieliset polut ovat useimmiten helpompia ymmärtää, käsitellä, ja muistaa.
- Mikäli tällä hetkellä selaajien saavuttamattomissa oleva sivu halutaan myöhemmässä vaiheessa sittenkin näkyviin julkiseen URL-osoitteeseen, sille on jo val-

miiksi varattu osoite. Mikään muu sivu ei voi, edes vahingossa, sijaita toiselle sivulle varatussa osoitteessa.

Koska sivutyypin asetuksissa voidaan määrittää mitä sivutyyppejä noudattavia sivuja sitä hyödyntävien sivujen alasivuiksi on mahdollista lisätä, tällä tavoin saadaan aikaan hallittuja sisältöhierarkioita. Sivutyypin asetuksista voidaan myös määrätä, ettei kyseistä sivutyyppiä noudattavilla sivuilla saa olla laisinkaan alasivuja – tietyissä tilanteissa varsin hyödyllinen rajaus, joka auttaa pitämään sivuston rakenteen loogisena ja eheänä.

1.3.2 Sivupohjat (templates)

Kukin sivu siis hyödyntää jotain järjestelmään luoduista sivupohjista, ja sivupohjat puolestaan muodostuvat kentistä. Yhtenä ProcessWiren kantavista ajatuksista onkin ollut se, että kaikki tietotyypit ovat kehittäjän (tai sivuston ylläpitäjän) määriteltävissä. Sivupohjia voidaan luoda järjestelmään haluttu määrän suoraan järjestelmän tarjoamin graafisin työkaluin, ja jokaiseen niistä voi liittää halutun määrän kenttiä.

1.3.3 Kentät (fields)

Kenttä on tietyn kenttätyypin nimetty ilmentymä, jolla on omat asetuksensa, ja joka mahdollistaa sisällön tallentamisen tiettyä lomakekenttää hyödyntäen. Kentät sidotaan sivupohjiin, mutta tietokantatasolla tallennettu tieto liittyy aina tiettyyn sivuun. Mikäli kenttä title on liitetty sivupohjaan basic-page, jokaisella tätä sivupohjaa hyödyntävällä sivulla on kyseinen kenttä, ja jokaisella niistä voi olla tässä kentässä oma arvonsa.

Kenttien luonti perustuu samaan ajatukseen kuin sivupohjienkin, eli näitä voidaan luoda käyttäjän toimesta haluttu määrä. Toisien kuin sivupohjien tapauksessa, kenttien osalta pohjaksi valitaan kuitenkin aina jokin valmis kenttätyyppi, jota kyseinen kenttä edustaa.

1.3.4 Kenttätyypit (fieldtypes)

Sisäänrakennettuja kenttätyyppejä ProcessWire sisältää oletuksena reilut kymmenen kappaletta, ja näitä ovat esimerkiksi Image (kuva), File (tiedosto), Text (teksti), Textarea (pitkä teksti), Page (sivu, voi sisältää viittauksia yhteen tai useampaan muuhun sivuun), Integer (kokonaisluku), sekä Email (sähköpostiosoite).

Koska kenttätyypit ovat moduuleja, niitä voidaan tarvittaessa lisätä järjestelmään vapaasti. Olennaisinta kenttätyypeissä on se, että jokainen niistä määrittää oman tietokantaskeemansa. Tähän sisältyvät niin tietorakenteet kuin niihin liittyvät indeksitkin. Kenttätyypit sisältävät vaihtelevassa määrin sovelluslogiikkaa liittyen arvojen ohjelmalliseen käsittelyyn tallennus- ja noutovaiheissa.

Sen lisäksi, että kenttätyyppi mahdollistaa tietyn tyyppisen tiedon tallentamisen, se myös määrittää, mitä lomakekenttiä sitä edustavat kentät voivat hyödyntää.

1.3.5 Lomakekentät (inputfields)

Lomakekentät ovat erillisiä moduuleja, ja pääpiirteissään niitä voi kuvata kenttätyypin sivuston hallinnassa näkyviksi ilmentymiksi, tai rajapinnoiksi sisällöntuottajien suuntaan. Kenttätyyppi määrittelee tietorakenteen, kun taas lomakekenttä määrittää sen, miten kyseisen tietorakenteen muokkaamiseen vaadittavat toiminnot esitetään ylläpitäjille.

Tärkein syy siihen, että kenttätyypit ja lomakekentät on erotettu toisistaan, on se, että näin yksittäisen kenttätyypin määrittämän sisällön muokkaaminen voidaan mahdollistaa useilla eri tavoilla. Esimerkiksi Page-kenttätyypin lomakekentäksi voidaan valita InputfieldSelect eli pudotusvalikko, mutta myös InputfieldCheckbox eli ruksivalinta – tai vaikkapa InputfieldPage, jolloin kohdesivu valitaan sivupuusta.

Toinen syy kenttätyyppien ja lomakekenttien erottamiselle on se, että siinä missä kenttätyyppi huolehtii viimekädessä sisällön tallennuksesta, lomakekentän tasolla puolestaan voidaan vaikuttaa siihen, millaisia valintoja käyttäjä voi tehdä. Lomakekenttä siis päättää, millaista sisältöä kyseiseen kenttään voidaan käyttöliittymän kautta syöttää.

Laadunvarmistuksen näkökulmasta kentän tehtävänä on kertoa käyttäjälle, mikäli jokin meni vikaan, eikä tarjottu syöte kelpaakaan kyseiseen kenttään. Siinä missä kenttätyypit usein yksinkertaisesti hylkäävät rikkinäiseksi katsomansa tiedot, lomakekentät pyrkivät ohjaamaan käyttäjää virheilmoitusten ja muiden pehmeämpien menetelmien avulla.

1.4 Kehittäjärajapinta

ProcessWiren kehittäjärajapinta tunnetaan yksinkertaisesti lyhenteellä API, joka tulee sanoista Application Programming Interface. Kyse on tarkemmin sisäisestä rajapinnasta, sillä ProcessWiren API ei ole saatavilla järjestelmän ulkopuolelta. Sen sijaan esimerkiksi ytimeen kuuluvat komponentit, moduulit, ja sivuaihiot hyödyntävät sitä keskustellakseen keskenään ja toimittaakseen sisältöä loppukäyttäjän nähtäville.

ProcessWiren kehittäjärajapinta pyrkii mukailemaan jQuery-kirjaston syntaksia. Vaikka konteksti onkin varsin erilainen, yhdistäviä piirteitä ovat muun muassa metodien nimeäminen, valitsinten hyödyntäminen (selectors), komentojen ketjuttaminen (chaining), sekä sisältöelementtien välillä liikkuminen (traversing).

Esikuvansa tapaan ProcessWiren API on suunniteltu sulavaksi (fluent), millä viitataan lähinnä asioiden loogiseen nimeämiseen ja ketjuttamisen kautta

```
<?php
echo $pages->get("/sivu/alasivu/")->images->eq(0)->width(600)->url;
```

LISTAUS 2. Esimerkki kehittäjärajapinnan komentojen ketjuttamisesta.

1.4.1 API-muuttujat

ProcessWiren kehittäjärajapinta on pohjimmiltaan kokoelma muuttujia, jotka edustavat järjestelmän ydinkäsitteitä ja -komponentteja. Näiden niin kutsuttujen API-muuttujien avulla voidaan esimerkiksi käsitellä sivuille tallennettuja tietoja, etsiä sivuja erilaisten kriteerien avulla valitsimia hyödyntäen, muokata istunnon tietoja, sekä lukea ja GET- tai POST-muodossa toimitettuja tietoja.

Hyviä esimerkkejä API-muuttujista ovat `$page`, jonka avulla voidaan käsitellä senhetkisen sivun sisältöä, ja `$user`, joka mahdollistaa vastaavan pääsyn aktiivisen käyttäjän tietoihin. API-muuttujien määrä on lisääntynyt järjestelmän kehittyessä, ja ajankohtainen listaus näistä kaikista löytyykin parhaiten järjestelmän virallisista käyttöohjeista.

1.4.2 API-funktiot

API-muuttujien lisäksi ydinkoodin yhteydestä löytyy tiedosto nimeltä Functions.php, jossa määritellään kokoelma ns. API-funktioita. API-funktiot ovat sivuston sisällä aina saatavilla olevia PHP-funktioita, joiden suurin ero API-muuttujiin nähden on se, että ne eivät suoraan liity suoraan mihinkään tiettyyn järjestelmän ydintoimintoon, vaan ovat luonteeltaan yleishyödyllisiä apuvälineitä.

Esimerkin vuoksi mainittakoon wireSendFile(), joka toimittaa määritetyn tiedoston suoraan käyttäjän selaimelle, ja wireMail(), jonka tehtävänä on sähköpostiviestien ohjelmallisen lähettämisen helpottaminen. API-funktioiden skaala on varsin laaja, ja niiden määrä on hiljalleen kasvanut järjestelmän kehittyessä – eli sitä mukaa kun uusille aputoiminnoille on ilmennyt tarvetta.

1.4.3 Valitsimet (selectors)

ProcessWiren valitsimet muistuttavat konseptitasolla läheisesti jQueryn valitsimia, jotka puolestaan pohjautuvat hyvin pitkälti CSS:n valitsimiin. Siinä missä jQueryn valitsimet ovat tapa kohdistaa komentoja sivun DOM-rakenteesta löytyviin olioihin, ProcessWiren valitsimet puolestaan mahdollistavat yksittäisten sivuston sivujen noutamisen annettujen kriteerien perusteella.

Valitsin koostuu kolmesta osasta: ominaisuuden nimi, operaattori, ja arvo. Valitsimia voidaan yhdistää yhdeksi merkkijonoksi, jolloin ne erotellaan toisistaan pilkuilla. Termeillä "include", "sort", ja "limit" on erityismerkitys: include määrittää näytetäänkö myös piilotetut tai julkaisemattomat sivut, sort määrittää hakutulosten järjestyksen, ja limit puolestaan määrittää sen, montako hakutulosta kerralla näytetään.

Yhdestä tai useammasta valitsimesta koostuva merkkijono (selector string) voidaan antaa parametrina API-muuttuja $pages metodille find(), joka palauttaa kaikki kyseistä valitsinmerkkijonoa vastaavat sivut PageArray-olion muodossa. Valitsimia voidaan hyödyntää myös muissa tilanteissa, kuten esimerkiksi valmiiksi noudetun PageArray-olion tarkemmassa rajauksessa filter()-metodin avulla.

```
// haetaan nimen mukaan järjestettyinä viisi sivupohjaa basic
// käyttävää sivua, joiden otsikon alku on "Hello Wo"
$items = $pages->find("
    template=basic-page, title^='Hello Wo', limit=5, sort=name
");

// haetaan käänteisessä järjestyksessä sivut, joiden kentistä joko
// name tai title sisältävät tekstin "ello"
$items = $pages->find("name|title%='ello', sort=-sort");

// haetaan sivut, jotka löytyvät osoitteessa /gallery/ sijaitsevan
// sivun alta, ja joilla on yksi tai useampi kuva kentässä images;
// poikkeuksellisesti näytetään tuloksissa myös piilotetut sivut
$items = $pages->find("
    images.count>0, parent='/gallery/', include=hidden
");
```

KAAVIO 3. Esimerkkejä valitsimista.

Valitsimet paitsi helpottavat monimutkaistenkin kyselyiden laatimista, myös tekevät järjestelmän sisäisistä kyselyistä varsin turvallisia. SQL-injektioiden mahdollisuus on käytännössä olematon, sillä määrämuotoisten valitsinten suodattaminen koneellisesti on raakoihin SQL-kyselyihin verrattuna huomattavasti suoraviivaisempi toimenpide.

1.4.4 Koukut (hooks)

Koukut mahdollistavat järjestelmän olemassa olevien toimintojen muokkaamisen tai korvaamisen sekä kokonaan uusien toimintojen lisäämisen ohjelmallisesti järjestelmän suorituksen aikana. Koukut ovat saaneet nimensä siitä, että tässä konseptissa kehittäjä määrittelee ns. koukkumetodin (hook method), joka kiinnitetään johonkin järjestelmän valmiista, koukkuja tukevista metodeista (hookable method).

Koukkumetodia kiinnittäessään kehittäjä voi päättää, suoritetaanko koukkumetodi ennen vai jälkeen alkuperäisen metodin – vaiko kenties sen tilalla, jolloin koukkumetodi korvaa alkuperäisen metodin kokonaisuudessaan. Koukkuja tukevien metodien toimintaa voidaan siis muokata kolmella tapaa:

- Koukkumetodi voidaan suorittaa ennen alkuperäisen metodin suoritusta, mikä mahdollistaa alkuperäiselle metodille annettujen parametrien muokkaamisen ennen kuin ne pääsevät kyseisen metodin käsiteltäviksi (before hook).
- Koukkumetodi voidaan suorittaa alkuperäisen metodin tilalla, jolloin se voi korvata alkuperäisen kohdemetodin kokonaisuudessaan (replace hook).
- Koukkumetodi voidaan suorittaa alkuperäisen metodin jälkeen, jolloin voidaan muokata metodin paluuarvoa (after hook).

Kaikki edellä mainituista mahdollistavat myös täysin uusien, omien toimintojen laukaisemisen haluttujen kriteerien täyttyessä. Koukkumetodi voi esimerkiksi tarttua sivun tallentamiseen, tarkistaa tallennettavat tiedot, ja niiden osuessa kohdalleen lähettää sähköpostia ylläpidolle, luoda tallennettavan sivun tietojen perusteella uuden sivun, tai muuttaa nykyisen tai jonkin aivan muun sivun kenttien arvoja.

Metodin muuttaminen koukkuja tukevaksi edellyttää kolmen alaviivan lisäämistä metodin nimen alkuun. Tämä saa aikaan sen, että metodia kutsuttaessa sitä ei löydykään suoraan tällä nimellä, jolloin kutsu ohjataan PHP:n maagiselle __call()-metodille. Tässä tapauksessa kyseinen metodi tarkistaa, löytyykö kohdeoliolta pyydettyä metodia kolmella alaviivalla varustettuna, ja mikäli löytyy, suorittaa sen – sekä siihen mahdollisesti kiinnitetyt koukkumetodit.

__call()-metodin toteutus löytyy ydinkomponentista nimeltä Wire, josta kaikki muut järjestelmän ydinkomponentit ja moduulit lopulta periytyvät. Tämän ansiosta koukkuja voidaan kiinnittää paitsi moduuleissa, myös aihiotiedostoissa. Ainoa varsinainen rajoitus on se, ettei koukku voi tarttua jo suoritettuihin toimintoihin. Usein koukkujen kiinnitys tapahtuukin esimerkiksi autoload-tyyppisen moduulin init()-metodissa, jolloin järjestelmän käynnistys on vasta aluillaan.

Suorituskykysyistä __call()-menetelmän rinnalle on tuotu vaihtoehtoinen malli, jossa sen sijaan, että metodi olisi suoraan koukkuja tukeva, luokalla on erikseen sama metodi ilman edellä mainittuja kolmea alaviivaa. Kun tätä metodia kutsutaan, se tarkistaa ja suorittaa alaviivalliseen versioonsa kytketyt koukut itsenäisesti. Tämä koskee käytännössä vain sellaisia metodeja, jotka saatetaan yhden kyselyn aikana suorittaa jopa satoja tai tuhansia kertoja, eli kyse on selkeästi poikkeustapauksesta.

Koukkumetodi saa syötteekseen HookEvent-luokan instanssin. HookEvent-luokan lähdekoodi dokumentoi luokan toiminnan, mutta pääpiirteissään HookEvent on tapa päästä kiinni kohdemetodin parametreihin, paluuarvoihin, ja niin edelleen. Esimerkiksi after-koukkujen tapauksessa HookEvent-oliolta löytyy return-parametri, joka sisältää kohdemetodin paluuarvon, ja jonka muokkaaminen muuttaa alkuperäisen metodin paluuarvoa.

Oheinen ohjelmakoodi esittelee yksinkertaisen koukkumetodin kiinnittämistä käytännössä. Tässä tapauksessa koukkumetodi `hookPageRender()` suoritetaan kaikkien Page-luokan instanssien (sivujen) `render()`-metodin kutsumisen jälkeen, mutta kuitenkin ennen kuin kyseisen metodin palauttama syöte ehditään näyttää käyttäjälle.

```
<?php

wire()->addHookAfter(
    'Page::render',
    null,
    'hookPageRender'
);

function hookPageRender(HookEvent $event) {
    // korjataan kirjoitusvirhe sivun sisällöstä
    $event->return = str_replace(
        "Harley Davidson",
        "Harley-Davidson",
        $event->return
    );
}

// render-metodin kutsuminen palauttaa arvon,
// jota hookPageRender on jo ehtinyt muokata
echo $page->render();
```

LISTAUS 3. Esimerkki yksinkertaisesta koukkumetodista ja sen kiinnittämisestä.

Eräs käytännön esimerkki kokonaan uusien toimintojen lisäämisestä on juuri edellä mainittu Page-luokan `render()`-metodi. Lähdekoodia tarkasteltaessa paljastuu, että Page-luokka ei itse asiassa sisällä lainkaan kyseistä metodia, vaan se lisätään järjestelmän suorituksen aikana erillisen PageRender-moduulin toimesta.

1.4.5 Tietokantayhteydet

Sivuston tai sovelluksen tarvitsee hyvin harvoin olla suoraan yhteydessä tietokantaan, sillä lähes kaikki tietojen hakuun tai tallentamiseen liittyvät toimenpiteet onnistuvat kehittäjärajapinnan ja valitsimien avulla. Tämä on yleensä hyvä idea jo siitäkin syystä, että näin vältetään myös vaarallisten – ja valitettavan yleisten – SQL-injektioiden riski.

Mikäli tietokantaan kaikesta huolimatta on päästävä suoraan kiinni, tähän tarkoitukseen voidaan hyödyntää ProcessWiren API-muuttujaa $database. Tämä on käytännössä PHP:n PDO-rajapinnan ympärille toteutettu suhteellisen kevyt kääreluokka, jonka ProcessWire alustaa automaattisesti järjestelmän käynnistyksen yhteydessä.

```
<?php
$stmt = $database->prepare("
    SELECT *
    FROM pages
    WHERE id = :pages_id
");
$stmt->bindValue(':pages_id', 1, PDO::PARAM_INT);
$stmt->execute();
$result = $stmt->fetch(PDO::FETCH_ASSOC);
echo "Etusivun on luonut " . $result['created_user'];
```

LISTAUS 4. Esimerkki $database-API-muuttujan hyödyntämisestä käytännössä.

Versiosta 2.4 lähtien ProcessWiren tietokantakyselyt ovat hyödyntäneet PDO-rajapintaa MySQLi-laajennuksen sijaan (Cramer 2/2014). PDO:n etuja ovat muun muassa tuki useille tietokantamoottoreille, nimetyille parametreille, sekä valmistelluille kyselyille. (Tatroe, MacIntyre, ja Lerdorf 2013: 205-206.)

Oikein hyödynnettynä PDO:n valmistellut kyselyt yhdistettynä parametrien tyypin määrittelyyn tekevät siitä myös MySQLi-laajennusta turvallisemman valinnan (Tatroe, MacIntyre, ja Lerdorf 2013: 292-294.) Kaikesta huolimatta suorien tietokantayhteyksien yhteydessä riski uusien haavoittuvuuksien ilmenemiselle korostuu, joten näitä tulisi välttää aina kun tämä mitenkään on mahdollista.

1.5 Yhteenveto ja pohdinta

ProcessWire on yhdistelmä sisällönhallintakehystä ja sisällönhallintajärjestelmää. Sen valttikortteja ovat näkymien ja tietorakenteiden muokattavuus, sekä kehittäjärajapinnan ja ylläpitotyökalujen käyttäjälähtöisyys. Kilpailevista tuotteista sen erottaa ennen kaikkea kyky suoriutua niin pienten sivustojen ja sovellusten kuin laajojen ja tietointensiivisten palveluidenkin tarpeista yksinkertaisuudesta ja helppokäyttöisyydestä tinkimättä.

Mikään ProcessWiren piirteistä ei ole sinänsä ennenkuulumaton, ja järjestelmän suunnittelussa onkin hyödynnetty surutta muista järjestelmistä lainattuja ideoita. Ainutlaatuiseksi ProcessWiren tekee kuitenkin se tapa, jolla kaikki tämä nivoutuu yhteen: yksikään järjestelmän piirteistä ei tunnu päälle liimatulta, vaan jokainen mukaan valittu palanen palvelee toinen toistaan ja sopii aukottomasti järjestelmän arkkitehtuuriin.

Tässä luvussa esitellyistä piirteistä osaa käsittelemme vielä myöhemmissä luvuissa tarkemmin. Kaikkeen ei kuitenkaan ole mahdollista pureutua kovin syvällisesti, sillä tämän kokoisessa järjestelmässä riittää jo melkoisesti asiaa käsiteltäväksi. Järjestelmän virallinen sivusto pitää sisällään laajasti tietoa muun muassa järjestelmän historiasta, erityispiirteistä, sekä yksittäisten ominaisuuksien toiminnasta, ja onkin kaikille ProcessWiresta kiinnostuneille erinomainen tietolähde.

2 SIVUPOHJAISTEN TIETORAKENTEIDEN PERUSTEET

Sisällönhallintakehyksenä eräs ProcessWiren tärkeimmistä tehtävistä on sisältötyyppien ja tietorakenteiden määrittelyn ja hallinnoinnin mahdollistaminen. Tässä luvussa perehdymme kyseisiin toimenpiteisiin sekä teorian että käytännön esimerkkien kautta. Tämän luvun ensisijaisena tavoitteena on tarjota perustason ymmärrys siitä, mitä ovat sivut, sivupohjat, ja sivupuu – ja miten ne liittyvät toisiinsa.

2.1 Sivut ja sivupohjat yleisellä tasolla

ProcessWiressa sisältö muodostuu sivuista, joista jokainen hyödyntää yhtä sivuston sivupohjista. Sivupohjat koostuvat kentistä, joita voidaan lisätä tai poistaa tarpeen mukaan. Kukin kenttä puolestaan noudattaa tiettyä kenttätyyppiä, joka määrittelee esimerkiksi sen, minkä tyyppistä tietoa kyseiseen kenttään voidaan tallentaa – kuvia, tekstiä, karttakoordinaatteja, viittauksia muihin sivuihin, ja niin edelleen.

Sekä kentät että sivupohjat ovat monikäyttöisiä, eli yleensä yhtä sivupohjaa hyödyntää useampi kuin yksi sivu, ja vastaavasti kaikki kentät sijaitsevat omassa "laarissaan", ja kukin niistä voidaan liittää niin moneen sivupohjaan kuin on tarvetta. Arkkitehtuurin kannalta olisi varmasti ollut mahdollista toteuttaa järjestelmä myös siten, että kentät ovat kiinteä osa sivupohjia, mutta monikäyttöisyydelläkin on omat etunsa.

Kullakin kentällä on oma tietokantataulunsa, jonka rakenteen sanelee kentän tyyppi. Esimerkiksi MapMarker-kenttätyyppiä hyödyntävien kenttien tietokantatauluilla on aina kentät "pages_id", "data", "lat", "lng", "status", ja "zoom" varsinaisen tiedon tallennusta varten. Kenttäkohtaisten tietokantataulujen etuna on se, että kunkin sivun kentän sisältö voidaan noutaa tietokannasta vain silloin, kun sille on todellista tarvetta.

Kunkin sivun kentät riippuvat siis täysin kyseisen sivun hyödyntämästä sivupohjasta. Mikäli sivulle halutaan esimerkiksi lisätä uusi kenttä, voidaan tätä tarkoitusta varten luoda uusi kenttä ja liittää se sivun hyödyntämään sivupohjaan – tai vaihtoehtoisesti liittää samaiseen sivupohjaan jokin jo aiemmin luoduista kentistä, mikäli se sopii tähän tarkoitukseen.

Tietorakenteiden kannalta äärimmäisen tärkeä sivupohjien ominaisuus ovat niin kutsutut perheasetukset (family settings), jotka mahdollistavat tietyntyyppisten sisältöjen niputtamisen yhteen. Niiden avulla voidaan esimerkiksi määritellä, että tuote-sivupohjaa hyödyntävä sivu voi sijaita vain tuotelistaus-sivupohjaa hyödyntävän sivun alla, ja sen alasivuina voi olla vain tuotevariaatio-sivupohjaa hyödyntäviä sivuja.

Tiettyä sivua ei ole sidottu pysyvästi tiettyyn sivupohjaan, vaan sen sivupohja voidaan jälkikäteen vaihtaa. Tällöin on kuitenkin syytä varmistua ensin siitä, ettei tietoa kadoteta lopullisesti – näin voi käydä lähinnä siinä tapauksessa, että vanhassa sivupohjassa oli jokin sellainen kenttä, jota uudessa ei enää ole. Järjestelmä pyrkii toki parhaansa mukaan varoittamaan käyttäjää erikseen tällaisissa riskialttiissa tilanteissa.

2.2 Sivupuu ja sivukohtaiset URL-osoitteet

Sivut noudattavat puumaista rakennemallia. Tässä mallissa kaikkein ylimmällä tasolla on aina vain ja ainoastaan yksi sivu, jolla puolestaan voi olla yksi tai useampi alasivu, joista kullakin voi olla omat alasivunsa, ja niin edelleen. Puumainen rakennemalli siis alkaa aina etu- tai juurisivusta ja haarautuu siitä eteenpäin juuri niin laajaksi ja syväksi kuin kyseisen sivuston tapauksessa on tarpeellista.

Jokaisella sivulla, oli kyseinen sivu julkisesti nähtävillä tai ei, on tässä mallissa aina oma URL-osoitteensa. Etusivun URL-osoite on "/", muiden sivujen osoitteet määräytyvät niiden nimien sekä yläpuolella olevien sivujen nimien mukaan: mikäli etusivun alla olevan sivun nimi on "yritys", sen URL-osoite on "/yritys/", ja mikäli sen alla puolestaan on sivu nimeltä "historia", sen URL-osoitteeksi tulee "/yritys/historia/".

```
Etusivu (/)
├── Tuotteet ja palvelut (/tuotteet-ja-palvelut/)
├── Yhteystiedot (/yhteystiedot/)
|   └── Palaute (/yhteystiedot/palaute/)
└── Yritys (/yritys/)
    ├── Henkilöstö (/yritys/henkilosto/)
    └── Historia (/yritys/historia/)
```

KAAVIO 4. Esimerkki URL-osoitteiden muodostumisesta sivupuussa.

Monella ProcessWire-sivustolla kullakin sivulla on vähintään kaksi sivun nimeen viittaavaa tietuetta: name ja title, eli vapaasti käännettyinä nimi ja otsikko. Sivun nimi on tietokantatasolta lähtien kiinteä ja pakollinen tietue, kun taas otsikko on mahdollista poistaa tai nimetä uudelleen, joskin useimmiten se jätetään käytännön syistä ennalleen.

- Sivun nimi (name) on pakollinen tietue, joka määrittää sivun URL-osoitteen viimeisen osan ja toimii eräänlaisena tunnisteena kyseiselle sivulle. Tämän kentän arvo on aina kirjoitettu pienin kirjaimin ja voi sisältää vain kirjaimia, numeroita, viivoja, ja alaviivoja.
- Sivun otsikko (title) on selkokielinen versio sivun nimestä. Otsikkoa käytetään usein esimerkiksi sivun pääotsikkona (h1) ja sivun nimenä valikoissa. Se voi sisältää vapaamuotoista tekstiä, ja kun sivu ensimmäisen kerran luodaan, sivun nimi (ja siten myös URL-osoite) muodostetaan automaattisesti sen otsikosta.

2.3 Virtuaaliset URL-osoitteet eli URL-segmentit

Yleissääntönä jokaisella sivulla on yksi osoite, ja jokainen osoite viittaa yhteen sivuun. Poikkeus tähän sääntöön ovat URL-segmentit, jotka mahdollistavat yhden sivun sijaitsemisen useassa erillisessä URL-osoitteessa – mutta kuitenkin vain yhden juurisivun alla. ProcessWiren URL-segmentit voidaankin mieltää virtuaalisiksi URL-osoitteiksi.

URL-segmenttien avulla esimerkiksi osoitteisiin /kartta/suomi/ ja /kartta/ruotsi/ saapuneet kyselyt voidaan ohjata käsiteltäviksi sivulle /kartta/, jolloin osoitteen loppuosa välitetään kyseiselle sivulle parametrin `$input->urlSegment1` arvona. Sovelluslogiikka ja merkkausta tuottava osuus säilyvät samoina, vaikka selaajan näkökulmasta kyse onkin kahdesta erillisestä sivusta.

Toinen yleisesti käytetty, URL-segmentteihin nojaava ratkaisu on yhden sivun esittäminen useissa sijainneissa. Esimerkiksi sivu /varaosat/moottori/knucklehead/ voi löytyä myös osoitteesta /varaosat/harley-davidson/1942/moottori/knucklehead/, mikä mahdollistaa sivujen luokittelun eri tavoin tilanteesta riippuen. Tällöin on toki huolehdittava myös siitä, että esimerkiksi hakukoneet tietävät mikä on sivun virallinen osoite.

Käytännössä kaikki URL-segmenttien avulla tehtävät asiat on mahdollista toteuttaa myös GET-parametrien avulla (/kartta/?alue=eurooppa). URL-segmenttien hyödyntä-

minen on usein joko visuaalinen- tai käytettävyyskysymys. Lisäksi URL-segmenttien yhteydessä voidaan hyödyntää järjestelmän omia sivutason välimuistiratkaisuja, mikä tekee niistä suorituskykynäkökulmasta GET-parametreja paremman vaihtoehdon.

URL-segmentit ovat sivupohjan asetuksista erikseen käyttöön otettava toiminto. Näiden asetusten avulla voidaan paitsi määrittää, montako tasoa keinotekoisia osoitteita sallitaan, myös nimetä erikseen kaikki sallitut virtuaaliset osoitteet. Tällä tavoin voidaan esimerkiksi sallia sivupohjalle kartta vain URL-segmentit suomi ja ruotsi, ja näyttää muissa tapauksissa (/kartta/eu/, /kartta/betelgeuse/, jne.) automaattisesti 404-virhe.

2.4 Sivujen väliset relaatiot eli sivuliitokset

Monipuolisten tietorakenteiden määrittelyssä relaatiot, eli yksittäisten tietueiden väliset liitokset, ovat erittäin tärkeässä asemassa. ProcessWiressa sivut edustavat tietueita ja näin ollen tietueiden väliset liitokset – eli relaatiot – ovat luonnollisesti sivujen välisiä liitoksia, eli sivuliitoksia.

Sivuliitosten määrittely onnistuu kenttien avulla. Tätä tarkoitusta varten on olemassa kenttätyyppi nimeltä "sivu", ja kuhunkin sivu-tyyppiseen kenttään voidaan kyseisen kentän asetuksista riippuen tallentaa viittaus yhteen tai useampaan sivuun. Muut sivukenttätyypin asetukset muun muassa määrittävät, millaisia sivuja kenttään voidaan valita (sivupohja, sijainti sivupuussa, jne.) sekä millaisen valintakentän avulla ne valitaan (ruksikenttä, pudotusvalikko, ennustava haku, jne.)

Teknisestä toteutustavastaan johtuen ProcessWiren sivuliitokset ovat yksisuuntaisia. Mikäli esimerkiksi sivulla A on sivu-tyyppinen kenttä, johon on valittu sivu B, vain sivu A tietää tästä liitoksesta. Mikäli liitoksesta halutaan kaksisuuntainen, pitää myös sivulla B valita sivu A vastaavaan sivu-tyyppiseen kenttään.

Yksinkertaisena esimerkkinä sivuliitosten mahdollisuuksista sivupohjalla "blogikirjoitus" voisi olla sivu-tyyppinen kenttä "kategoriat", johon voidaan valita yksi tai useampi "kategoria"-sivupohjaa noudattava sivu. Vastaavasti sivupohjalla "tuote" voisi olla sivutyyppiset kentät "tuoteryhma", "veroluokka", sekä "hyllypaikka".

2.5 Teoriasta käytäntöön: yksinkertaisen uutissivuston rakenne

Perusteiden läpikäynnin jälkeen voimmekin perehtyä tietorakenteiden käsittelyyn käytännön esimerkin kautta. Esimerkkinä käytämme uutissivustoa, joka esimerkillisestä yksinkertaisuudestaan huolimatta perustuu aiempiin kokemuksiin vastaavan kaltaisten sivustojen pystyttämisestä ja on näin ollen kaikin puolin täysin toteuttamis- ja käyttökelpoinen ratkaisu.

Tämän osion tavoitteena ei kuitenkaan ole esitellä parhaita ratkaisuja, vaan ennen kaikkea auttaa hahmottamaan, miten sivujen ja niiden välisten liitosten avulla voidaan toteuttaa monipuolisesti erilaisia tietorakenteita. Jokainen sivusto on omanlaisensa, ja koska ProcessWire mahdollistaa sisältörakenteiden määrittelyn tehokkaasti ja nopeasti, tätä mahdollisuutta kannattaa myös hyödyntää ahkerasti.

2.5.1 Tietotyyppien määrittely

Uutta sivustoa suunniteltaessa on suositeltavaa lähteä liikkeelle aivan alusta, eli tarpeellisten tietotyyppien määrittelystä. Tähän työvaiheeseen kannattaa uhrata riittävästi aikaa, sillä tietotyypit muodostavat ProcessWire-sivuston perustan. Kun tietotyypit on määritelty, niiden keskinäiset suhteet ja sijainti sivupuussa usein loksahtavat ikään kuin itsestään kohdalleen.

Koska esimerkissämme on kyse uutissivustosta, ensimmäinen mieleen tuleva tietotyyppi on todennäköisesti uutinen, eli "news-item". Sivurakenteen kannalta ei ole erityisen tärkeää, millaisia kenttiä uutisella on, mutta koska varsinaisen sivuston kannalta tällä sen sijaan on hyvinkin paljon merkitystä, kaavio 5 esittää yhtä mahdollista tapaa näiden organisointiin.

```
- Perustiedot
    - Otsikko (title, FieldtypePageTitle)
    - Julkaistaan alkaen (publish_from, FieldtypeDatetime)
    - Piilotetaan alkaen (publish_until, FieldtypeDatetime)
    - Ingressi (summary, FieldtypeTextarea)
    - Pääsisältö (body, FieldtypeTextarea)
    - Kuvat (images, FieldtypeImage)
    - Liitetiedostot (files, FieldtypeFile)
```

```
- Metatiedot
    - Kirjoittaja (author, FieldtypePage)
    - Asiasanat (meta_keywords, FieldtypeTextarea)
    - Kuvaus (meta_description, FieldtypeTextarea)
```

KAAVIO 5. Sivupohjan "news-item" kentät.

Kunkin tietotyypin – kuten tässä tapauksessa uutisten – luokittelu omiin osioihinsa auttaa selkeyttämään sivuston rakennetta, ja usein sille on myös muita perusteluja, kuten esimerkiksi yhteisen koontisivun mahdollistaminen. Tätä tarkoitusta varten luomme uutisille oman pääsivunsa, jota varten määrittelemme uuden sivupohjan nimeltä "news".

```
- Perustiedot
    - Otsikko (title, FieldtypePageTitle)
```

KAAVIO 6. Sivupohjan "news" kentät.

Uutisten lisäksi sivustomme kaipaa tapaa luokitella uutiset – esimerkiksi asiasanojen tai kategorioiden avulla. Nämä eivät välttämättä ole toisensa poissulkevia vaihtoehtoja, mutta tässä esimerkissä valitsemme kuitenkin lähestymistavaksi kategoriat. Tätä varten tarvitsemme uuden sivupohjan nimeltä "category", jonka kentät kuvataan kaaviossa 7.

```
- Perustiedot
    - Otsikko (title, FieldtypePageTitle)
```

KAAVIO 7. Sivupohjan "category" kentät.

Myös kategoriat haluamme omaan osioonsa, joten luomme niille oman pääsivunsa. Tätä varten tarvitsemme jälleen yhden sivupohjan, "categories", jonka kentät on kuvattu kaaviossa 8.

```
- Perustiedot
    - Otsikko (title, FieldtypePageTitle)
```

KAAVIO 8. sivupohjan "categories" kentät.

Viimeinen tietotyyppi, jonka sivustomme tarvitsee, edustaa uutisten kirjoittajia, ja on nimeltään "author". Periaatteessa voisimme hyödyntää suoraan kunkin uutisen luoneen käyttäjän tietoja, mutta tällöin uutisartikkelin kirjoittajaa ei olisi mahdollista vaihtaa, eikä yhdellä uutisella voisi koskaan olla useampaa kuin yhtä kirjoittajaa. Kaavio 9 kuvaa kirjoittaja- eli author-sivupohjan tärkeimpiä kenttiä.

```
- Perustiedot
    - Otsikko (title, FieldtypePageTitle)
    - Etunimi (name_first, FieldtypeText)
    - Sukunimi (name_last, FieldtypeText)
    - Kuva (image, FieldtypeImage)
```

KAAVIO 9. sivupohjan "author" kentät.

Aiempien tietotyyppien tapaan luomme myös kirjoittajia varten oman kokoomasivunsa, ja tätä varten sivupohjan "authors". Kaavio 10 kuvaa kyseisen sivupohjan kenttiä.

```
- Perustiedot
    - Otsikko (title, FieldtypePageTitle)
```

KAAVIO 10. sivupohjan "authors" kentät.

Tarkkasilmäinen lukija varmasti ihmettelee viimeistään tässä vaiheessa, miksi loimme useita sivupohjia täsmälleen samoilla kentillä. Vastaus on yksinkertaisesti se, että näin mahdollistetaan perheasetusten hyödyntäminen (categories-sivupohjaa käyttävän sivun alla voi olla vain category-sivupohjaa käyttäviä sivuja, ja niin edelleen). Lisäksi kunkin tietotyypin kokoomasivulle (categories, news, authors) voidaan näin myöhemmin toteuttaa oma, muista kokoomasivuista kokonaan riippumaton näkymänsä.

2.5.2 Sivurakenteen hahmottelua

ProcessWire on joustava järjestelmä, ja tietyn asian toteuttamiseen on hyvin harvoin olemassa vain yksi ainoa oikea tapa. Yleensä vaihtoehtoja on useita, eikä valinta niiden väliltä ole aina täysin itsestään selvä. Toisaalta mikään vaihtoehto tuskin on yksiselitteisesti väärä, sillä kaikilla ratkaisuilla on aina omat hyvät ja huonot puolensa.

Kaavio 11 kuvaa rakennemallia, jossa sivujen ja kategorioiden välinen suhde perustuu puhtaasti hierarkiaan, eli sivujen sijaintiin sivupuussa. Tämä malli on monessa tilanteessa oikein toimiva ratkaisu, ja vähintäänkin äärimmäisen selkeä ja helppo ymmärtää.

```
Etusivu
└── Uutiset
|   ├── Kotimaa
|   |   ├── Uutinen 1
|   |   ├── Uutinen 2
|   |   └── Uutinen 3
|   ├── Ulkomaat
|   |   ├── Uutinen 4
|   |   └── Uutinen 5
|   └── Talous
|       └── Uutinen 6
└── Kirjoittajat
    ├── Kirjoittaja 1
    ├── Kirjoittaja 2
    └── Kirjoittaja 3
```

KAAVIO 11. esimerkki yksinkertaisesta, puumaisesta sivuhierarkiasta, jossa yksi uutinen kuuluu aina yhteen kategoriaan.

Yllä näkyvässä, sivuhierarkiaan perustuvassa mallissa on kuitenkin varsin merkittävä rajoite – nimittäin se, että tällöin yksi uutinen voi aina kuulua vain yhteen kategoriaan. Riippuu täysin tilanteesta, onko tällä merkitystä vai ei, mutta esimerkkisivustollamme hierarkiaan perustuva suhde ei tunnu mielekkäältä ratkaisulta. Entäpä jos haluammekin eritellä kotimaata koskevat talousuutiset ulkomaita koskevista talousuutisista?

Jotta uutisen kuuluminen useampaan kategoriaan voidaan mahdollistaa, rakennemallia täytyy miettiä hieman toisesta näkökulmasta. Kaavion 12 esimerkki kuvaa sivurakennetta, jossa uutiset ja kategoriat on tätä tarkoitusta silmällä pitäen erotettu toisistaan ja niiden väliset yhteydet toteutetaan sivuhierarkian sijaan sivuliitoksia hyödyntäen.

```
Etusivu
└── Uutiset
|   ├── Uutinen 1
|   ├── Uutinen 2
|   ├── Uutinen 3
```

```
|   ├── Uutinen 4
|   ├── Uutinen 5
|   └── Uutinen 6
└── Kategoriat
|   ├── Ulkomaat
|   |   ├── Aasia
|   |   ├── Afrikka
|   |   └── Eurooppa
|   ├── Kotimaa
|   └── Talous
└── Kirjoittajat
    ├── Kirjoittaja 1
    ├── Kirjoittaja 2
    └── Kirjoittaja 3
```

KAAVIO 12. esimerkki sivuliitoksiin perustuvasta sivuhierarkiasta.

Sivuliitosten mahdollistamiseksi joudumme muuttamaan hieman sivupohjalle "uutinen" aiemmin määrittelemäämme tietorakennetta. Muutos on pieni, mutta merkittävä: lisäämme sivu-tyyppisen kentän nimeltä "kategoriat", jotta voimme liittää uutisen juuri niin moneen kategoriaan kuin kulloinkin on tarpeen.

```
- Perustiedot
    - Otsikko (title, FieldtypePageTitle)
    - Julkaistaan alkaen (publish_from, FieldtypeDatetime)
    - Piilotetaan alkaen (publish_until, FieldtypeDatetime)
    - Ingressi (summary, FieldtypeTextarea)
    - Pääsisältö (body, FieldtypeTextarea)
    - Kuvat (images, FieldtypeImage)
    - Liitetiedostot (files, FieldtypeFile)
- Metatiedot
    - Kirjoittaja (author, FieldtypePage)
    - Kategoriat (categories, FieldtypePage)
    - Asiasanat (meta_keywords, FieldtypeTextarea)
    - Kuvaus (meta_description, FieldtypeTextarea)
```

KAAVIO 13. sivupohjan "news-item" uusi, sivuliitoksia varten sovitettu tietorakenne.

Kuten jo aiemmin kerroimme, sivuliitoksilla viitataan tapaan, jolla ProcessWiren kenttätyyppiä "sivu" noudattava kenttä voi sisältää viittauksen yhteen tai useampaan muuhun sivuun. Sivuliitoksen avulla tehty viittaus mahdollistaa aihiotiedostossa viitattavan

sivun tietojen tulostamisen suoraan viittauksen kautta. Kentän arvon pyytäminen palauttaa Page-luokkaa edustavan olion, eli sivun.

```
<div itemprop=author itemscope itemtype="http://schema.org/Person">
    <span itemprop="name">
    <?= $page->author->name_first ?> <?= $page->author->name_last ?>
    </span>
</div>
```

LISTAUS 14. sivu-tyyppiseen kenttään tallennetun sivuviittauksen hyödyntäminen aihiotasolla.

Mikäli kyseinen sivu-tyyppinen kenttä mahdollistaa useamman sivun valitsemisen, siihen tallennetun arvon hyödyntäminen tapahtuu hieman eri tavoin. Tällöin ProcessWire palauttaa arvoa pyydettäessä yksittäisen sivun sijaan iteroinnin mahdollistavan PageArray-olion, joka puolestaan koostuu useista erillisistä sivuista.

```
<?php if (count($page>categories)): ?>
<div itemprop='keywords'>
    <?php $max = count($page->categories); ?>
    <?php $num = 0; ?>
    <?php foreach ($page->categories as $category): ?>
        <?php $num = $num + 1; ?>
        <?= $category->title ?>
        <?php if ($num < $max): ?>, <?php endif; ?>
    <?php endforeach; ?>
</div>
<?php endif; ?>
```

LISTAUS 15. Sivu- kenttään tallennettujen viittausten hyödyntäminen aihiotasolla, kun kenttä sallii useamman kuin yhden viittauksen tallentamisen samanaikaisesti.

PageArray-olioita voidaan siis iteroida samaan tapaan kuin PHP:n omia array-muuttujiakin. PageArray mahdollistaa myös tällaisiin tilanteisiin hyvin sopivan oikotien, joka muistuttaa jälleen PHP:n omia toimintoja: PHP:n implode() -funktio mahdollistaa arrayn sisältämien tietueiden "liimaamisen" yhteen määriteltyä erotinmerkkiä hyödyntäen, ja sama toiminto löytyy myös PageArray-olioilta:

```
<?php if (count($page>categories)): ?>
<div itemprop='keywords'>
    <?= $page->categories->implode(", ", "title") ?>
</div>
<?php endif; ?>
```

LISTAUS 16. edellisen listauksen toteutus hyödyntäen PageArray-olion implode() -metodia: erottimena käytetään pilkkua ja kustakin sivusta näytetään kentän "title" arvo.

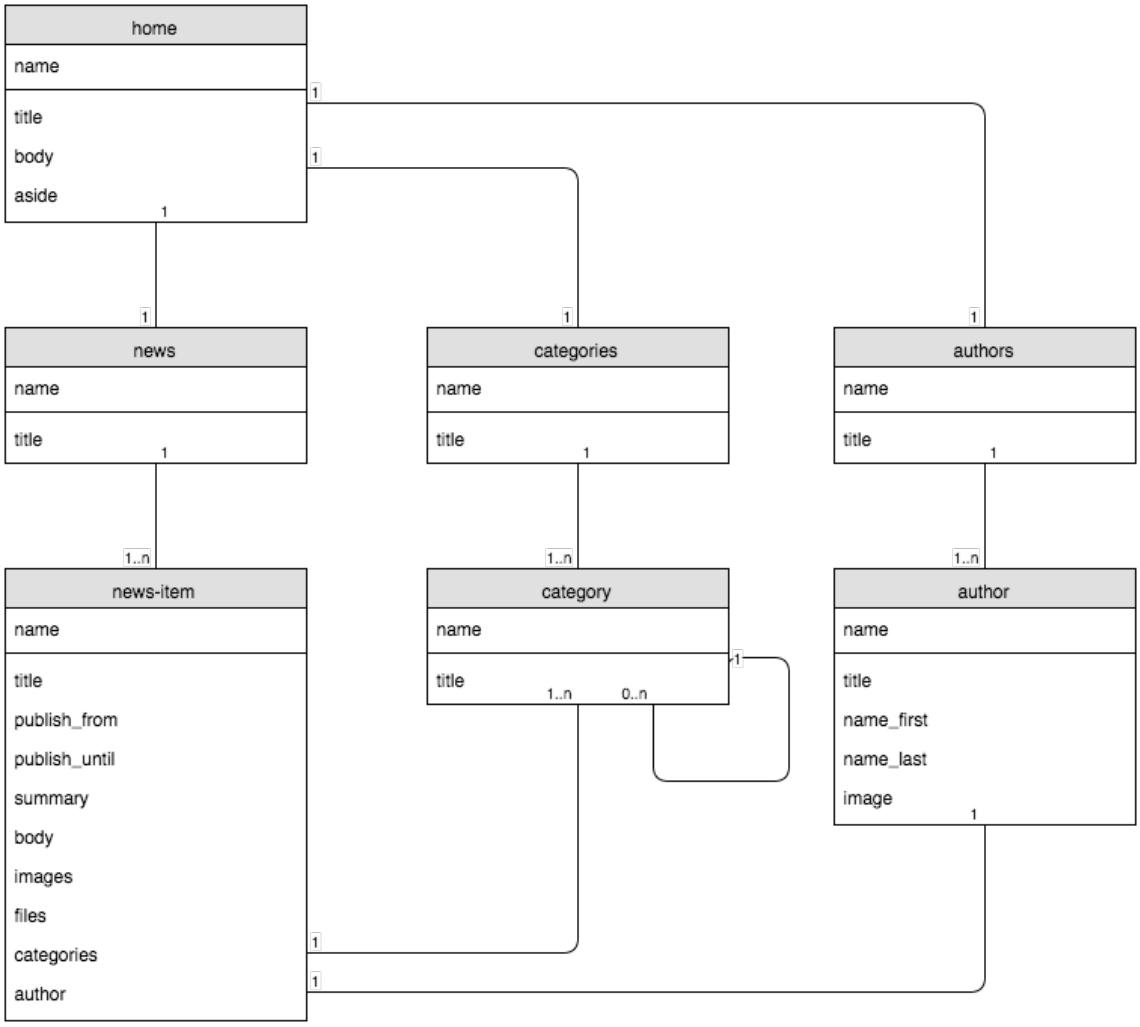
2.5.3 Lyhyt kertaus ja yhteenveto

Tässä luvussa esittelimme täysin käyttökelpoisen uutissivuston tietorakenteen. Tässä esimerkissä hyödynnettiin ensisijaisesti sivuliitoksia erilaisten tietotyyppien välisten suhteiden, eli relaatioiden, mahdollistamiseen:

- Uutisten käyttämään sivupohjaan ("news-item") on lisätty sivu-tyyppinen kenttä "categories", johon voidaan valita esimerkiksi pudotuslistasta haluttu määrä "category"-sivupohjaa noudattavia sivuja, eli kategorioita.
- Uutisen (myöskin sivu-tyyppiseen) kenttään "author" voidaan valita jokin "author"-sivupohjaa noudattavista sivuista, eli uutisten kirjoittajista. Kirjoittajilla on muun muassa omat nimitietonsa sekä oma profiilikuvansa, jotka voidaan esittää uutista selaavalle loppukäyttäjälle halutussa muodossa.

Myös sivujen välistä hierarkiaa hyödynnetään, joskin pääasiassa tietotyyppien luokitteluun: uutisten ("news-item") ylätasona toimii uutislistaus ("news"), kategorioiden ("category") ylätasona toimii kategorialistaus ("categories"), ja niin edelleen.

Kaavio 17 kuvaa sivustomme tietorakenteita UML:n keinoin. Kuten kaaviosta näemme, esimerkkirakenteemme sisältää kaikesta yksinkertaisuudestaan huolimatta jo varsin monenlaisia liitoksia.



KAAVIO 17. Sivuliitoksiin perustuvan uutissivuston sivupohjien keskinäiset suhteet sekä tärkeimmät kentät UML-kaaviona.

Aihiotiedostoissa sivuliitoksia voidaan hyödyntää esimerkiksi siten, että uutista luettaessa näytetään suoraan linkit kaikkiin kategorioihin, joihin se on liitoksissa. Vastaavasti Kategoriat-sivulla voidaan ensin hakea kaikki sivun alasivut, eli yksittäiset kategoriat, ja tämän jälkeen käydä ne läpi yksi kerrallaan, samalla listaten kunkin kategorian kohdalla ne uutiset, jotka viittaavat kyseiseen kategoriaan.

Sivuliitokset vaativat aihiotasolla hieman enemmän sovelluslogiikkaa kuin edellä mainittu "yksi uutinen kuuluu yhteen kategoriaan" -tyyppinen rakenne, mutta toisaalta ne ovat myös tuovat todella merkittävää joustoa sivurakenteiden suunnitteluun.

2.6 Yhteenveto ja pohdinta

Järjestelmään ensimmäistä kertaa tutustuville ProcessWiren terminologia sekä omintakeinen tapa jäsentää asioita saattaa aiheuttaa päänvaivaa. Ensimmäinen haaste onkin siis sen tiedostaminen, että sivu ei suinkaan tarkoita etukäteen määriteltyä tietotyyppiä, vaan kunkin sivun tietotyypin määrittää sen hyödyntämä sivupohja. Sivupohjat koostuvat kentistä, jotka ovat monikäyttöisiä: yksi kenttä voi liittyä moneen eri sivupohjaan.

Myös sivupohjien ja kenttien toteutustapa voi vaikuttaa rajoittuneelta, sillä järjestelmä ei varsinaisesti suosi kertakäyttöisiä, vain yhtä käyttötapausta varten luotuja sivupohjia tai kenttiä. Pienen totuttelun jälkeen ratkaisu käy kuitenkin järkeen: paitsi että kyse on tietokantatason optimoinnista, monikäyttöisyydestä seuraa myös se käytännön etu, että tällöin muutosten tekeminen sivupohjiin tai kenttiin jälkikäteen onnistuu keskitetysti.

Sen lisäksi, että tietotyypit ovat vapaasti määriteltävissä, ProcessWire tarjoaa työkalut myös sivujen välisten relaatioiden eli sivuliitosten hallinnointiin. Sivuliitokset mahdollistavat minkä tahansa sivun liittymisen mihin tahansa muuhun sivuun aina kun tälle on tarvetta, ja ne taipuvat niin kategorioiden, tuoteryhmien, kohderyhmien, veroluokkien, kuin referenssienkin tarpeisiin, täysin tilanteesta riippuen.

Tietorakenteiden määrittely on selkeästi ProcessWiren vahvuus. Tämä ei tule varsinaisesti yllätyksenä, sillä kyse on kuitenkin sisällönhallintakehyksestä, ja tietorakenteiden määrittely liittyy sisällönhallintaan olennaisesti. Tapa jolla ProcessWire tätä tehtäväänsä hoitaa on kuitenkin ainutlaatuisen yksinkertainen, ja tästä syystä sitä voi hyvällä omallatunnolla suositella taustajärjestelmäksi niin monipuolisille ja tietointensiivisille sovelluksille kuin kaikenkokoisille sivustoillekin.

3 SIVUAIHIOIDEN TOTEUTTAMISEN PERUSTEET

ProcessWiren sivupohjat (templates) määrittävät, mitkä kentät kyseistä sivupohjaa hyödyntävillä sivuilla on, missä sivut voivat sijaita, kenellä on oikeus niiden muokkaamiseen ja katseluun, ja niin edelleen. Kuhunkin sivupohjaan voi liittyä myös aihiotiedosto (template file), joka puolestaan määrittää sivupohjaa hyödyntävien sivujen esitystavan, eli esimerkiksi HTML-rakenteen ja sen sekaan tulostettavat tietokentät.

Jokaisella sivulla on oma URL-osoitteensa, joka perustuu sen nimeen ja sijaintiin sivupuussa. Mikäli sivun nimi on ”ota-yhteytta”, ja se sijaitsee sivun ”yhteystiedot” alla, joka puolestaan sijaitsee sivuston etusivun alla, sivun URL-osoite on /yhteystiedot/ota-yhteytta/. Mikäli kyseisen sivun hyödyntämään sivupohjaan liittyy aihiotiedosto, eivätkä sivupohjan oikeusrajaukset tätä estä, sivu on myös selaajien nähtävillä tässä polussa.

Sen sijaan sivu, jonka sivupohjaan ei liity aihiotiedostoa, ei voi olla selaajien, koska sille ei tällöin ole määritelty lainkaan esitystapaa. ProcessWire ei siis koskaan määritä sivupohjalle automaattisesti esitystapaa, vaan tämä on kehittäjän tehtävä. Sivupohja voidaan kuitenkin asettaa hyödyntämään samaa aihiotiedostoa toisen sivupohjan kanssa.

3.1 Aihiotiedostojen nimeäminen ja sijainti

Aihiotiedostot sijaitsevat hakemiston /site/templates/ alla sivupohjan mukaan nimettyinä. Esimerkiksi sivupohjan basic-page aihiotiedosto on /site/templates/basic-page.php, sivupohjan home aihiotiedosto /site/templates/home.php, ja niin edelleen. Ainoa poikkeus tähän sääntöön ovat ne sivupohjat, jotka on edellä kuvatusti asetettu hyödyntämään muuta kuin oletusaihiotiedostoa. Tällöin aihiotiedoston nimi voidaan määrittää vapaasti.

Oletuksena aihiotiedostojen päätteenä käytetään PHP-kielen .php-päätettä. Päätteen voi tarvittaessa asettaa myös joksikin muuksi muokkaamalla sivuston asetustiedostoa /site/config.php. Toisinaan näkee käytettävän esimerkiksi .phtml-tiedostopäätettä, mutta mitään todellista hyötyä tällä ei saavuteta – oikeastaan päinvastoin, sillä web-palvelin ei välttämättä oleta näin nimetyn tiedoston sisältävän PHP-sovelluskoodia, mistä puolestaan voi seurata erinäisiä tietoturvaongelmia (Shiflett 2006: 51).

Hieman myöhemmin tässä opinnäytteessä esittelemme mallin, jossa sivupohjien käyttö on viety muutamaa askelta pidemmälle, ja mukaan on tuotu muun muassa erillinen front controller -kerros, eli esikäsittelijä. Tässä vaiheessa riittää kuitenkin tieto siitä, missä sivupohjat sijaitsevat, ja miten niiden nimi oletusarvoisesti määräytyy.

3.2 Aihiotiedostojen rakenne ja sisältö

ProcessWiren aihiotiedostot ovat tavallisia PHP-tiedostoja, ja voivat siten sisältää esimerkiksi HTML-merkkausta ja PHP-sovelluskoodia rinnakkain. Sovelluskoodi tosin on tässä kohtaa hieman yliampuva sanavalinta, sillä useimmiten tällä tarkoitetaan lähinnä muutamia `echo`-, `if`-, ja `foreach`-käskyjä, eikä juuri mitään sen monimutkaisempaa.

Erillisten moduulien avulla aihiotiedostoissa voidaan hyödyntää PHP:n sijaan esimerkiksi Twig-, Smarty-, tai Latte-aihiokieliä. Erityisesti aihiointitarkoitukseen suunniteltuilla kielillä on omat hyvät ja huonot puolensa, mutta näiden tarkempi käsittely ei sisälly tämän opinnäytteen aihepiiriin.

HTML:n sijaan sivupohja voi esittää sivun sisällön myös esimerkiksi XML-, JSON-, tai YAML-muodossa. ProcessWire ei ota kantaa siihen, millaisen sisällön esittämiseen sitä hyödynnetään, ja kykenee siten toimimaan tavanomaisen, selaimella tarkasteltavan websivuston sijaan vaikkapa JavaScript-sovelluksen taustajärjestelmänä, eli rajapintana ja abstraktiotasona front-end -toteutuksen ja tietokannan välillä.

Esimerkin vuoksi oletetaan kuitenkin, että olemme toteuttamassa perinteistä websivustoa, haluamme tulostaa HTML-merkkausta, ja tarkoitus olisi näyttää selaajille sisältö sivun "title" (tekstimuotoinen otsikko) ja "body" (useimmiten HTML-muotoinen pääpalsta). Aihiotiedostossa tilanne näyttäisi tällöin jokseenkin tältä:

```
<h1><?php echo $page->title; ?></h1>
<div id="body"><?php echo $page->body; ?></div>
```

LISTAUS 5. Kenttien arvojen tulostaminen aihiotiedostossa echo-lausekkeilla API-muuttujaa `$page` hyödyntäen.

Esimerkissä hyödynnetään API-muuttujaa $page sivun kenttien arvojen noutamiseen. API-muuttujia käsittelimme luvussa 3, mutta kertauksen vuoksi mainittakoon, että $page on yksi ProcessWiren automaattisesti aihiotiedostojen käyttöön alustamista muuttujista, ja edustaa nykyistä sivua kenttineen ja metodeineen, eli valmiine toimintoineen.

Koska PHP on varsin "aihioystävällinen" kieli, esimerkiksi eräs sen yleisimmin käytetyistä ominaisuuksista, sisällön tulostaminen echo-lausekkeen avulla, on mahdollista esittää myös lyhyemmässä muodossa <?= ... ?>. Alla näkyvä esimerkki vastaa toiminnaltaan edellistä esimerkkiä, mutta hyödyntää lyhyitä echo-lausekkeita:

```
<h1><?= $page->title ?></h1>
<div id="body"><?= $page->body ?></div>
```

LISTAUS 6. Lyhyen echo-lausekkeen hyödyntäminen.

Kuvitellaanpa tilannetta, jossa sivulla on kentät title ja headline. Mikäli headline-kenttään on syötetty arvo, halutaan se esittää sivulla, mutta jos kenttä on jätetty tyhjäksi, halutaan sen sijaan näyttää title-kenttään tallennettu arvo. ProcessWiresta löytyy tähän tarpeeseen sisäänrakennettu toiminnallisuus: $page-get("kentta1|kentta2|kentta3").

```
<h1><?= $page->get("headline|title") ?></h1>
<div id="body"><?= $page->body ?></div>
```

LISTAUS 7. Vaihtoehtoisten kenttien hyödyntäminen get()-metodin avulla.

Tässä esimerkissä käytetty get()-metodi on muutoin vastaava kuin ns. nuolinotaatio ($page->kentta), ja sitä voidaan käyttää myös yksittäisen arvon noutamiseen. Useimmiten nuolinotaatio on kuitenkin yksittäisten arvojen kohdalla selkeämpi vaihtoehto.

```
<h1><?= $page->get("headline|title") ?></h1>
<div id="body"><?= $page->body ?></div>
<?php if ($page->sidebar) { ?>
    <div id="sidebar"><?= $page->sidebar ?></div>
<?php } ?>
```

LISTAUS 8. Ehtolausekkeen hyödyntäminen aihiotiedostossa.

Listauksessa 8 tarkistetaan if- eli ehtolausekkeen avulla, onko sivupalkissa (sidebar) sisältöä, ja mikäli on, tulostetaan kentän sisältö pääpalstan jälkeen. PHP:ssä kaikki yleisimmät kontrollirakenteet (if, while, for, foreach ja switch) voidaan esittää myös vaihtoehtoisessa, erityisesti aihioiden näkökulmasta selkeämmässä, muodossa. Alla vastaava toteutus vaihtoehtoisella if..endif -ehtorakenteella:

```
<h1><?= $page->get("headline|title") ?></h1>
<div id="body"><?= $page->body ?></div>
<?php if ($page->sidebar): ?>
    <div id="sidebar"><?= $page->sidebar ?></div>
<?php endif; ?>
```

LISTAUS 9. Ehtolausekkeen hyödyntäminen aihiotiedostossa hyödyntäen vaihtoehtoista merkintätapaa PHP:n kontrollirakenteille.

ProcessWire tarjoaa kehittäjän käyttöön myös toistettavia sisältöjä. Näissä tapauksissa kentän arvo ei enää olekaan yksittäinen merkkijono, vaan kokoelma erillisiä arvoja tai olioita. Esimerkiksi kuvakentässä voi, kentän asetuksista riippuen, olla useita kuvia, jolloin niiden esittäminen sivustolla vaatii foreach-lausekkeen hyödyntämistä:

```
<h1><?= $page->get("headline|title") ?></h1>
<div id="body"><?= $page->body ?></div>
<?php if ($page->sidebar): ?>
    <div id="sidebar"><?= $page->sidebar ?></div>
<?php endif; ?>
<div id="images">
    <?php foreach ($page->images as $image): ?>
        <img src="<?= $image->url ?>"
             alt="<?= $image->description ?>" />
    <?php endforeach; ?>
</div>
```

LISTAUS 10. Foreach-silmukkarakenteen hyödyntäminen aihiotiedostossa.

Otetaan mukaan vielä hieman laadunvarmistusta, eli lisätään tarkistus sille, oliko kuvia oikeasti olemassa, ennen kuin ne sisältävä elementti tuodaan näkyviin. Tässä esimerkissä käytetään PHP:n count()-funktiota kuvakentän kuvien lukumäärän laskemiseen:

```
<h1><?= $page->get("headline|title") ?></h1>
<div id="body"><?= $page->body ?></div>
<?php if ($page->sidebar): ?>
    <div id="sidebar"><?= $page->sidebar ?></div>
<?php endif; ?>
<?php if (count($page->images)): ?>
    <div id="images">
        <?php foreach ($page->images as $image): ?>
            <img src="<?= $image->url ?>"
                 alt="<?= $image->description ?>" />
        <?php endforeach; ?>
    </div>
<?php endif; ?>
```

LISTAUS 11. PHP-kielen toimintojen hyödyntäminen aihiotiedostossa.

Viimeinen aihiotason toiminnallisuus, jonka käymme vielä lyhyesti läpi ennen siirtymistä seuraavaan asiaan, on `include`. Tämä PHP:n ominaisuus mahdollistaa muualla sijaitsevan tiedoston noutamisen suoritettavan tiedoston sisään, ja on erittäin käytännöllinen aihiotiedostojen pilkkomisessa uudelleenkäytettäviin, pienempiin paloihin.

Oheisessa esimerkissä määrittelemme ylä- ja alapalkin, joita hyödynnämme varsinaisissa aihiotiedostoissa. Include-tiedostomme on nimetty hieman aihiotiedostoista poiketen, eli nimen alkuun on lisätty alaviiva. Tämä on yksi konventio, mutta aivan yhtä hyvin alaviivan voisi jättää myös pois, tai tiedostot voisi esimerkiksi siirtää eri hakemistoon.

Kun aihiotiedostot pilkotaan tällä tavoin osiin, ne pysyvät selkeämpinä ja muutosten hallinta helpottuu. Mikäli esimerkiksi sivuston head-osioon on tarpeen tehdä muutoksia, nämä tehdään yhteen paikkaan jokaisen aihiotiedoston sijaan. Include-lausekkeet sivupohjakohtaisissa aihiotiedostoissa ovat helppo tapa aihiotiedoston pilkkomiseen:

```
<!DOCTYPE html>
<html>
    <head>
        <title><?= $page->title ?></title>
    </head>
    <body>
```

LISTAUS 12. Header-tiedosto, eli sivuston yläpalkki (_header.php).

```
        <p>This site is proudly powered by ProcessWire CMS/CMF</p>
    </body>
</html>
```

LISTAUS 13. Footer-tiedosto, eli sivuston alapalkki (_footer.php).

```
<?php include './_header.php' ?>
    <h1><?= $page->get("headline|title") ?></h1>
    <div id="body"><?= $page->body ?></div>
    <?php if ($page->sidebar): ?>
        <div id="sidebar"><?= $page->sidebar ?></div>
    <?php endif; ?>
    <?php if (count($page->images)): ?>
        <div id="images">
            <?php foreach ($page->images as $image): ?>
                <img src="<?= $image->url ?>"
                     alt="<?= $image->description ?>" />
            <?php endforeach; ?>
        </div>
    <?php endif; ?>
<?php include './_footer.php' ?>
```

LISTAUS 14. valmis aihiotiedosto kokonaisuudessaan.

Include-lausekkeiden suhteen on kuitenkin syytä olla varovainen, ja esimerkiksi käyttäjältä saadun syötteen hyödyntäminen tässä yhteydessä on usein huono idea; vähintäänkin ensin pitää varmistua siitä, että käyttäjän antama tiedoston nimi on sellainen, joka todella voidaan tähän kohtaan noutaa. (Shiflett 2006: 53-55.)

Mikäli tässä kohtaa tuntuu siltä, että uutta asiaa on tullut paljon lyhyessä ajassa, tästä ei kannata turhaan huolestua – yllä näkyvä esimerkki nimittäin sisältää käytännössä jo kaikki yleisimmät sivuaihioissa tarvittavat ominaisuudet:

- Sisällön tulostamisen (`echo` tai `<?= ... ?>`)
- Ehtolausekkeiden hyödyntämisen (`if..endif`)
- PHP-kielen toimintojen hyödyntämisen (`count()`)
- Silmukoiden hyödyntämisen (`foreach..endforeach`)
- Aihion pilkkomisen pienempiin kokonaisuuksiin (`include`)

Kehittäjärajapinnan toiminnot mahdollistavat huomattavasti monipuolisemmatkin toteutukset, mutta tässä luvussa esiteltyjen menetelmien hyödyntäminen ja soveltaminen riittää jo monen yksinkertaisen sivuston tarpeisiin. Esimerkit ovat tarkoituksellisesti varsin karsittuja, eikä niissä ole merkkausta juuri nimeksikään; todellisuudessa tilanne on usein tältä osin varsin toisenlainen.

3.3 Kuvien käsittely aihiotiedostoissa

Välimuistiratkaisujen yhteydessä kerrottiin, että ProcessWire luo automaattisesti sisältökuvista oikeat kokovariaatiot, ja tallentaa ne välimuistikopioina levylle. Kokovariaatioita voi syntyä joko silloin, kun kuvia upotetaan RTE-kentän (esimerkiksi CKEditor) avulla sivuston sisältöön, mutta tietty variaatio voidaan luoda myös ohjelmallisesti aihiotiedostossa.

Kun kuvakentän arvoa kysytään aihiossa, vastauksena on joko Pageimages-olio, joka sisältää useita kuvatiedostoja (kts. aiemmat esimerkkimme toistettavista elementeistä), tai vaihtoehtoisesti Pageimage-olio, joka edustaa yhtä kuvatiedostoa. Pageimage-oliolla on tiettyjä erityisesti kuviin liittyviä metodeja, joista size(), width(), ja height() mahdollistavat kuvan skaalaamisen ohjelmallisesti.

```
<!-- Kuvatiedosto 240x160 pikselin koossa -->
<img src="<?= $page->image->size(240, 160)->url ?>" alt="" />

<!-- Kuvatiedosto 240 pikseliä leveänä (korkeus suhteellinen) -->
<img src="<?= $page->image->width(240)->url ?>" alt="" />

<!-- Kuvatiedosto 160 pikseliä korkeana (leveys suhteellinen) -->
<img src="<?= $page->image->height(240)->url ?>" alt="" />
```

LISTAUS 15. Kuvien skaalaaminen ohjelmallisesti tiettyyn kokoon aihiotiedostossa size()-metodin avulla.

Pageimage-olion size() -metodille voidaan antaa myös kolmas parametri (ja width() sekä height() -metodeille toinen parametri), array-muuttuja $options. Tämän muuttujan avulla voidaan vaikuttaa seuraaviin kuvan skaalaukseen liittyviin asetuksiin:

- **quality=90**, uuden kuvan laadun asettaminen asteikolla 1-100
- **upscaling=true**, sallitaanko alkuperäistä suuremman kuvan luonti?
- **cropping=center**, leikkauksen keskipiste; useita mahdollisia asetuksia
- **suffix='word'**, kuvan nimeen lisättävä teksti; yksittäinen sana tai array
- **forceNew=true**, pakotetaanko joka kyselyllä kuvan luonti uudelleen?
- **sharpening=soft**, terävöitys; vaihtoehdot none, soft, medium ja strong
- **autoRotation=true**, käännetäänkö kuva automaattisesti pystysuuntaan?
- **rotate=0**, kuvan kääntäminen asteina: 0, -270, -180, -90, 90, 180 tai 270
- **flip=''**, peilikuva; vaihtoehdot joko vertical (vaaka) tai horizontal (pysty)
- **hidpi=false**, käytetäänkö pikseleiden tuplausta hidpi-/retina-näytöille?
- **cleanFilename=false**, siivotaanko nimestä aiempien variaatioiden tiedot?

```
<img src="<?= $page->image->size(240, 160, array(
    'upscaling' => false, // korkeintaan alkuperäisen kuvan koko
    'quality' => 75, // kuvan laatu 75
))->url ?>" alt="" />
```

LISTAUS 16. Kuvan skaalaaminen ohjelmallisesti tietyin asetuksin.

3.4 API-muuttujat ja -funktiot

API-muuttujat edustavat ProcessWiren kehittäjärajapintaa ja avaavat pääsyn esimerkiksi nykyisen (selattavan) sivun sekä muiden sivujen tietoihin, nykyisen käyttäjän ja muiden sivuston käyttäjien tietoihin, sekä moneen muuhun tietueeseen. Niiden avulla on myös mahdollista ohjelmallisesti muuttaa, poistaa, tai lisätä järjestelmään tietoja.

Käsittelimme kehittäjärajapintaa lyhyesti ProcessWiren ydinkäsitteiden yhteydessä ja aihiotiedostoesimerkeissämme hyödynsimme myös API-muuttujaa nimeltä $page sivun kenttien arvojen noutamiseen. Alla vielä lyhyt kuvaus tärkeimmistä ja yleisimmin aihiotiedostoissa käytetyistä API-muuttujista:

- **$page** edustaa nykyistä sivua ja mahdollistaa pääsyn sen kenttiin:
    - Kentän arvo: `$page->title`, `$page->get('headline|title')`
    - Sivun näkyvyyden tarkistus: `if ($page->viewable()) { ... }`
    - Sivun muutosten tallennus: `$page->save()`, `$page->save('title')`

    - Sivun poistaminen: `$page->delete()`
- **$pages** edustaa kaikki järjestelmän sivuja:
    - Sivun noutaminen osoitteen, numeerisen tunnisteen tai valitsimen avulla: `$pages->get('/ajankohtaista/')`, `$pages->get(1)`
    - Sivujen noutaminen valitsimen avulla: `$pages->find('template=news-item, parent=/ajankohtaista/, sort=created, limit=5')`
    - Sivun kopiointi: `$pages->clone($page)`, `$pages->clone($page, $parent)`
- **$user** edustaa nykyistä käyttäjää:
    - Onko käyttäjä kirjautunut: `if ($user->isLoggedin()) { ... }`
    - Onko käyttäjällä tietty rooli: `if ($user->hasRole('superuser')) { ... }`
- **$input** mahdollistaa pääsyn POST- ja GET-parametreihin sekä evästeisiin:
    - GET-muuttujan "id" arvon lukeminen: `$input->get->id`
    - POST-muuttujan "name" arvon lukeminen: `$input->post->name`
    - Evästeen "wire" arvon lukeminen: `$input->cookie->wire`
- **$sanitizer** mahdollistaa sisällön siistimisen eli sanitoinnin:
    - Parametrin HTML-enkoodaus: `$sanitizer->entities($input->get->param)`
    - URL:n oikeellisuus: `if ($sanitizer->url('http://www.example.com')) { ... }`

API-muuttujien lisäksi ProcessWire tarjoaa käyttöön myös kokoelman ns. API-funktioita. Nämä ovat yleishyödyllisiä toimintoja, jotka eivät suoraan liity mihinkään järjestelmän valmiista olioista, mutta edustavat kuitenkin yleisesti käytettyjä toimintoja. Alla muutamia esimerkkejä ProcessWiren API-funktioista:

- **wireMail()** mahdollistaa sähköpostien lähettämisen ohjelmallisesti hyödyntäen järjestelmään kulloinkin asennettua WireMail-sähköpostimoduulia
- **wirePopulateStringTags()** korvaa tekstistä formaatin mukaiset "tagit" niitä vastaavilla muuttujilla; sekä teksti että muuttujat annetaan funktiolle parametreina
- **wireMkDir()** luo uuden, ProcessWire-järjestelmälle avoimen hakemiston
- **wireCopy()** kopioi kaikki tiedostot lähdehakemistosta kohdehakemistoon
- **wireZipFile()** luo hakemistosta tai yksittäisistä tiedostoista ZIP-paketin
- **wireUnzipFile()** purkaa valmiiksi luodun ZIP-paketin kohdehakemistoon
- **wireSendFile()** lähettää tiedoston HTTP-yhteyden yli suoraan selaimelle

API-muuttujia ja -funktioita on huomattava määrä ja kullakin niistä on omat ominaispiirteensä ja käyttötarkoituksensa. Tarkempaa tietoa ja ajankohtainen listaus kaikista

kehittäjien saatavilla olevista API-muuttujista ja -funktioista löytyy ProcessWire-järjestelmän virallisesta ohjeistosta.

3.5 Yhteenveto ja pohdinta

ProcessWiren oliopohjainen, ketjutusta tukeva kehittäjärajapinta tekee aihiotasolla sivun – ja koko sivuston – sisällön parissa toimimisesta vaivattoman ja miellyttävän kokemuksen. Rajapinnan oppimiskynnys on myös varsin alhainen, ja luonnollisesti kynnys madaltuu entisestään, mikäli käyttäjällä on valmiiksi kokemusta esimerkiksi PHP:n ja muiden sisällönhallintakehysten tai -järjestelmien parissa toimimisesta.

Se, mikä tekee ProcessWiren aihioista erityisen joustavia, on kuitenkin ennen kaikkea järjestelmän merkkausriippumattomuus. Kuten yllä jo mainittiinkin, sivuaihio voi HTML:n sijaan esittää sivun sisällön esimerkiksi XML- tai JSON-, YAML- tai PDF-muodossa. PHP:n keinoin kehittäjä voi lähettää haluamansa header-tietueet selaimelle, ja siten esimerkiksi kertoa sille, mitä tiedostomuotoa sen pyytämä sivu edustaa (Tatroe, MacIntyre, ja Lerdorf 2013: 182.).

Sivuaihioiden laatiminen on ehdottomasti niitä aiheita, joista voisi kirjoittaa laajemminkin. Tämän luvun tarkoituksena oli kuitenkin tärkeimpien kohtien nopea läpikäynti pääasiassa esimerkkien kautta, ja tämä tavoite myös täyttyi. Myöhemmissä luvuissa käsittelemme vielä sivuaihioiden hyödyntämistä hieman monimutkaisemmassa käyttötapauksessa MVC-mallin variaation kautta.

4 YKSINKERTAINEN MODEL-VIEW-CONTROLLER -MALLI

ProcessWire ei suosi tiettyä sisäänrakennettua arkkitehtuurimallia, vaan sen kantavana ajatuksena on aina ollut joustava rakenne, joka voidaan mukauttaa minkä tahansa arkkitehtuurimallin tarpeisiin – tai aivan yhtä hyvin olla mukauttamatta, mikä mahdollistaa esimerkiksi kaikkein yksinkertaisimpien sivustojen tai sovellusten laatimisen kokonaisuudessaan yhden ainoan aihiotiedoston sisään.

Joustavuuden seurauksena ProcessWire mahdollistaa lukemattomia erilaisia lähestymistapoja esimerkiksi nykyaikaisissa web-ratkaisuissa varsin suositun Model-View-Controller -mallin toteuttamiseen. Tässä luvussa esittelemmekin yhtä näistä hieman tarkemmin, joskin itse MVC-arkkitehtuurin käymme läpi melko tiiviinä pakettina, sillä sen laajempi käsittely ei kuulu tämän opinnäytteen aihepiiriin.

Ennen siirtymistä monipuolisempien rakennemallien pariin on kuitenkin syytä käsitellä muutaman kappaleen verran yksinkertaisempia malleja, sekä syitä niiden olemassaoloon. Tämä auttaa kokonaiskuvan hahmottamisessa, ja toivottavasti tarjoaa myös arvokasta näkemystä ProcessWire-järjestelmän suunnitteluperiaatteisiin.

4.1 Lähtökohtana yksinkertaisuus: Direct Output –malli

ProcessWiren oletussivupohjat perustuivat pitkään yksinkertaistettuun rakenteeseen, jossa kullakin sivupohjalla oli vain yksi aihiotiedosto. Kyseinen aihiotiedosto sisälsi sekä sovelluslogiikan, että näyttöpuolen, yhdeksi kokonaisuudeksi paketoituina:

```
<html>
    <head>
        <title><?= $page->title ?></title>
    </head>
    <body>
        <h1><?= $page->title ?></h1>
        <div id="body">
            <?= $page->body . $page->comments->render() ?>
        </div>
        <div id="sidebar">
            <?php
```

```
            if (count($page->children)) {
                echo "<ul id='menu'>";
                foreach ($page->children as $child) {
                    echo "<li>"
                       . "<a href='{$child->url}'>"
                       . "{$child->title}"
                       . "</a>"
                       . "</li>";
                }
                echo "</ul>";
            }
            ?>
            <?= $page->sidebar ?>
        </div>
    </body>
</html>
```

LISTAUS 17. basic-page.php (Direct Output)

Tässä mallissa kaikki sivupohjaan liittyvä löytyy yhdestä ja samasta tiedostosta, mikä on etenkin aloittelevan kehittäjän kannalta suoraviivaista, selkeää ja helposti omaksuttavaa. Järjestelmään "sisään pääseminen" haluttiin näin tehdä mahdollisimman yksinkertaiseksi – mitä se toki olikin, joskin tästä saattoi ajoittain aiheutua myös mielikuva, ettei järjestelmä taivu tämän monipuolisempaan malliin.

Yksinkertaisiin toteutuksiin tämä Direct Output -mallina tunnettu ratkaisu on toimiva ja riittävä, mutta etenkin laajemmissa ja monimutkaisemmissa ratkaisuissa kaivataan monipuolisempaa ja hienostuneempaa ratkaisua. Tästä syystä osa nykyisistä oletussivupohjista hyödyntää monipuolisempaa Delayed Output -mallia. (Cramer 2015b.)

4.2 Skaalautuva ja tehokas Delayed Output -malli

Delayed Output -mallissa sisällön osat valmistellaan etukäteen, minkä jälkeen ne koostetaan yhteen ja tulostetaan selaajan näkyville erillisessä näkymätiedostossa. Tässä mallissa sovelluslogiikka eli sisällön koostaminen ja esitystapa on erotettu toisistaan:

```
<?php
$title = $page->title;
$body = $page->body . $page->comments->render();
```

```
$sidebar = "";
if (count($page->children)) {
    $sidebar = "<ul id='menu'>";
    foreach ($page->children as $child) {
        $sidebar .= "<li><a href='{$child->url}'>"
                  . "{$child->title}</a></li>";
    }
    $sidebar .= "</ul>";
}
$sidebar .= $page->sidebar;
include "./main.inc";
```

LISTAUS 18. basic-page.php (Delayed Output)

```
<html>
    <head>
        <title><?= $title ?></title>
    </head>
    <body>
        <h1><?= $title ?></h1>
        <div id="body"><?= $body ?></div>
        <div id="sidebar"><?= $sidebar ?></div>
    </body>
</html>
```

LISTAUS 19. main.inc (Delayed Output)

Tämä malli tarjoaa tiettyjä, selkeitä etuja verrattuna aiemmin esiteltyyn Direct Output -malliin. Ensinnäkin sovelluslogiikka ja näkymä pysyvät erillään, mikä mahdollistaa näkymän muuttamisen sovelluslogiikkaan kajoamatta. Lisäksi näkymän eriyttäminen aihiotiedostosta mahdollistaa sen, että useampikin sivupohja voi hyödyntää samaa näkymää – Direct Output -mallissa näkymä oli aina tiukasti sidottu sivupohjan sovelluslogiikkaan, eikä siis hyödynnettävissä erillään.

Kaikkein monipuolisimmissa ja vaativimmissa tapauksissa tämäkään ei välttämättä riitä, vaan kaivataan erityisesti suurempien sovellusten yhteyteen suunniteltua rakennemallia. Eräs suosituimmista malleista tähän käyttöön on MVC, eli Model–View–Controller, jota käsittelemme seuraavaksi hieman tarkemmin.

4.3 Monipuolinen ja joustava MVC-malli

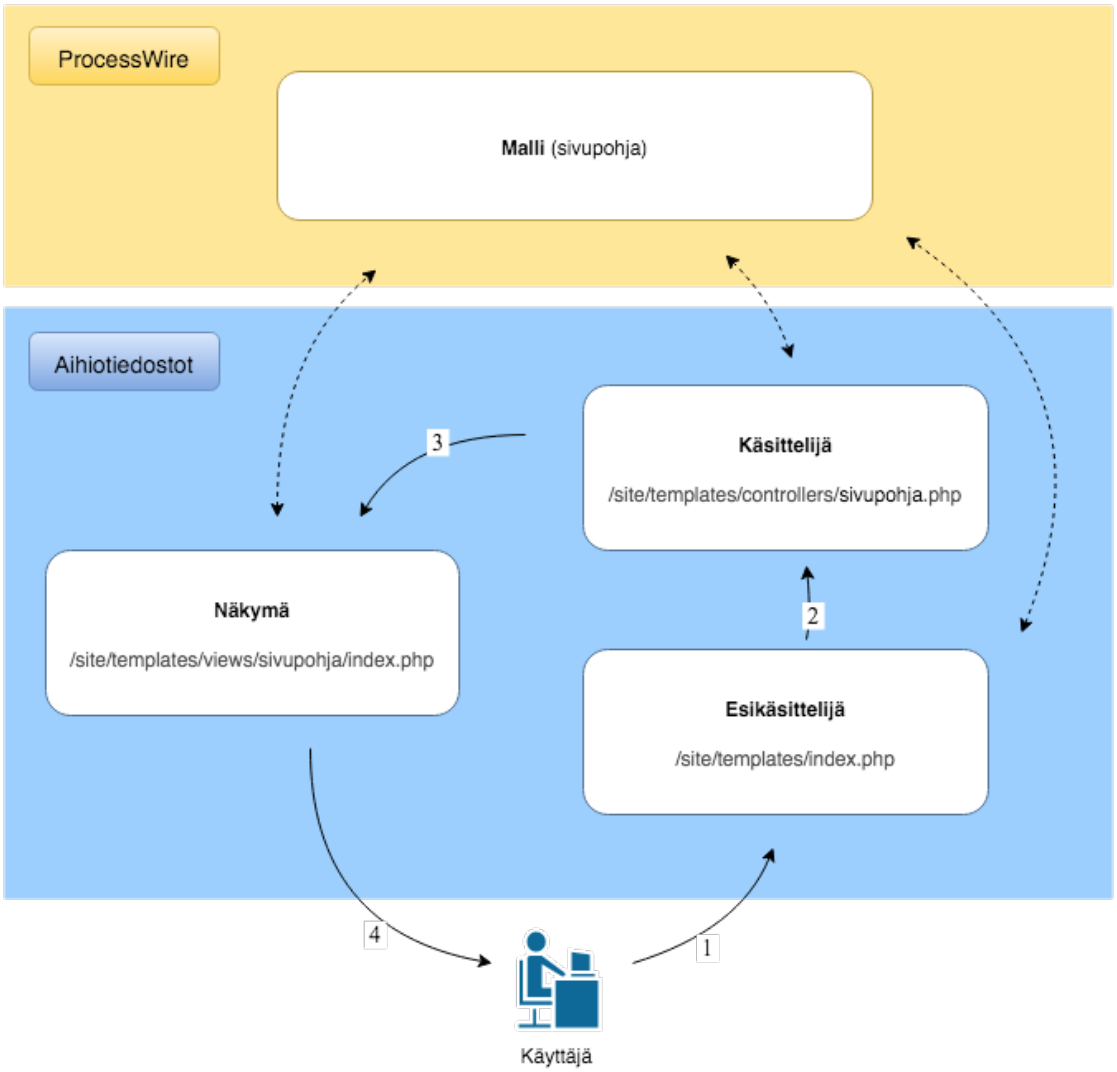
MVC (Model-View-Controller) on web-kehityksen parissa suosittu sovelluskehityksen arkkitehtuurimalli, jossa sovellus jaetaan kolmeen itsenäiseen, toisiaan täydentävään osaan: malliin (model), näkymään (view), ja käsittelijään (controller). Jokaisella näistä on oma roolinsa sovelluksen elinkaaressa. (Coggeshall 2009: 3-5).

- **Malli** (Model) kuvastaa ja hallinnoi sovelluksen tietorakenteita ja niihin liittyvää sovelluslogiikkaa. Kaikki tietorakenteisiin tai dataan tehtävät pyynnöt ja muutokset kulkevat malli-kerroksen kautta.
- **Näkymä** (View) vastaa käyttöliittymätarpeista, ja tarjoaa siten käyttäjälle konkreettisen kosketuspinnan sovellukseen. Näkymä ei yleensä sisällä omaa sovelluslogiikkaa, mutta voi pyytää tietoja suoraan malli-kerrokselta.
- **Käsittelijä** (Controller) vastaanottaa käyttäjän tekemät pyynnöt, tulkitsee ne, ja lopulta välittää ne eteenpäin mallille sekä päivittää niiden pohjalta näkymää.

MVC-malli sai alkunsa vuonna 1979, jolloin Trygve Reenskaug vierailevana tutkijana Xeroxin PARC-laboratoriossa kehitti Thing-Model-View-Editor -arkkitehtuurin monimutkaisten sovellusten tarpeisiin (Reenskaug 5/1979). Samana vuonna julkaistiin toinenkin dokumentti, joka määritteli terminologian osittain uudelleen; tässä yhteydessä käytettiin jo termejä Model, View ja Controller (Reenskaug 12/1979).

Vuosien varrella MVC-mallia on varioitu eri tarpeisiin, ja näin siitä onkin muodostunut yläkäsite joukolle varsin erilaisia käytännön toteutuksia. Tässä luvussa esiteltävä MVC-mallin mukaelma on sovellettu ProcessWiren sivupohjien kontekstiin ja – erinäisten muiden eroavaisuuksien ohella – muun muassa lisää malliin kokonaan uuden front controller -komponentin.

Front controller, esikäsittelijä, on sovelluskehityksen suunnittelumalli (pattern), jossa kaikki sovellukseen saapuvat pyynnöt ohjataan ensivaiheessa esikäsittelijälle, jonka tehtävänä puolestaan on siirtää ne käsiteltäviksi eteenpäin. Tämä malli mahdollistaa esimerkiksi kaikkia pyyntöjä koskevien yhteisten toimintojen toteuttamisen vaivattomasti. (Zandstra 2008: 238; Coggeshall 2009: 5-6).



KAAVIO 18. Elementit ja niiden väliset interaktiot opinnäytetyön aihiotasolla toteutettavassa MVC-mallin mukaelmassa. Katkoviivat kuvaavat mahdollista yhteyttä, nuolet viestien suuntaa. Nuoli molemmissa päissä viittaa kahdensuuntaiseen liikenteeseen.

4.3.1 Hakemistorakenne

Hieman alempaa löytyy hakemistorakenne, jota hyödynnämme tässä MVC-esimerkissämme. Hakemistorakenne on muokattu ja sovitettu ProcessWire-sivustolle, mutta sen lähtökohtana ja löyhänä esikuvana on hyödynnetty Zend Framework -sovelluskehyksen projektirunkoa.

Rakenne lähtee aihiohakemistosta (/site/templates/), jonka alla on omat hakemistonsa käsittelijöille (/site/templates/controllers/), näkymille (/site/templates/views/), sekä staattisille resursseille (/site/templates/static/). Lisäksi hakemistosta löytyvät myös yk-

sittäiset tiedostot index.php sekä admin.php, joista ensimmäinen on hyödyntämämme esikäsittelijä ja jälkimmäinen liittyy ProcessWiren ylläpitonäkymiin.

```
.
└── site
    └── templates
        ├── admin.php
        ├── index.php
        ├── controllers
        ├── errors
        ├── static
        |   ├── css
        |   ├── img
        |   └── js
        └── views
            ├── layouts
            |   └── default.php
            ├── partials
            |   └── menu
            |       └── top.php
            └── scripts
                ├── basic-page
                |   └── index.php
                └── home
                    ├── index.php
                    └── json.php
```

KAAVIO 19. MVC-esimerkkitoteutuksen hakemistorakenne.

Seuraavaksi käymme läpi rakentamamme MVC-kehyksen komponentit ja hakemistot yksi kerrallaan, aloittaen esikäsittelijästä ja päätyen lopulta näkymien kautta staattisiin resursseihin. Lopuksi kerromme vielä lyhyesti, miten aiemmin esitellyt komponentit toimivat yhdessä, ja miten tätä mallia voidaan tarvittaessa mukauttaa tai laajentaa.

4.3.2 Esikäsittelijä

Ensimmäisenä keskitymme esikäsittelijään, joka vastaanottaa kaikki sivustoon kohdistuvat kyselyt, käynnistää sivupohjakohtaisen käsittelijän, välittää parametrit käsittelijälle ja näkymälle – ja lopulta huolehtii sivun piirtämisestä. Esikäsittelijä on vapaa suomennos termille Front Controller, joka puolestaan on eräs yleisesti tunnetuista ohjelmointimalleista (Zandstra, 2008: 238).

Vaikka tässä esiteltävä esikäsittelijä onkin varsin kevyt variaatio varsinaisesta Front Controller -mallista, on se kaikesta huolimatta hyödyntämistämme komponenteista kenties monimutkaisin. Syynä tähän on se, että esikäsittelijän vastuulla on kaikkien sellaisten tehtävien suorittaminen, jotka muutoin jouduttaisiin suorittamaan erikseen jokaisessa käsittelijätiedostossa.

```
<?php

/**
 * Esikäsittelijä
 *
 * Esikäsittelijä toimii ensimmäisenä askeleena sivustoon
 * kohdistuville kyselyille sekä valitsee ja suorittaa
 * käsittelijän ja näkymätiedoston.
 *
 */

// alustetaan muuttujat
$ext = ".{$config->templateExtension}";
$views = "{$config->paths->templates}views/";
$scripts = "{$views}scripts/{$page->template}/";
$controllers = "{$config->paths->templates}controllers/";

// alustetaan näkymä
$view = new TemplateFile;
$view->set('layout', 'default');
$view->set('placeholders', new WireData);
$view->setGlobal('staticURL', "{$config->urls->templates}static/");

// haetaan näkymätiedostojen osat (partials) myöhempää käyttöä varten
// ja välitetään ne näkymätiedostolle muuttujana nimeltä 'partials'
function getPartials($path, $ext) {
    $partials = array();
    foreach (glob($path) as $partial) {
        $name = basename($partial);
        if (is_dir($partial)) {
            $partials[$name] = getPartials("{$partial}/*", $ext);
        } else if (strrpos($name,$ext)===strlen($name)-strlen($ext)) {
            $partials[substr($name,0,strrpos($name, "."))] = $partial;
        }
    }
    return (object) $partials;
}
$view->partials = getPartials("{$views}partials/*", $ext);
```

```
// ladataan sivupohjakohtainen käsittelijä; koska kyseessä ei ole
// pakollinen komponentti, tarkistetaan ensin löytyykö tiedostoa
if (is_file("{$controllers}{$page->template}{$ext}")) {
    include "{$controllers}{$page->template}{$ext}";
}

// valitaan näkymätiedosto; oletusarvon sijaan näkymän voi asettaa
// myös $page-olion parametrilla 'view' tai GET-parametrilla 'view'
if (!$view->filename ||
    !is_file("{$scripts}{$view->filename}{$ext}")) {
    $name = basename($input->get->view ?: ($page->view ?: 'index'));
    if (is_file("{$scripts}{$name}{$ext}")) {
        $view->filename = "{$scripts}{$name}{$ext}";
        if ($name != "index") {
            // muu kuin oletusnäkymä valittu, ohitetaan välimuisti
            $session->PageRenderNoCachePage = $page->id;
        }
    }
}

// mikäli näkymä- tai asettelutiedosto on määritelty, piirretään sivu
if ($view->filename || $view->layout) {
    $content = $view->render();
    if ($name = basename($view->layout)) {
        // asettelutiedosto (layout) mahdollistaa yhteisen sivurungon
        // jakamisen usean eri näkymätiedoston kesken (DRY-periaate)
        $layout = new TemplateFile;
        $layout->filename = "{$views}layouts/{$name}{$ext}";
        $layout->partials = $view->partials;
        $layout->placeholders = $view->placeholders;
        if (!$layout->placeholders->content) {
            $layout->placeholders->content = $content;
        }
        $content = $layout->render();
    }
    echo $content;
}
```

LISTAUS 20. MVC-mallin esikäsittelijä (/site/templates/index.php).

Sivupohjakohtaisia, valinnaisia käsittelijöitä, sekä erilaisia tapoja rakentaa näkymä (näkymä- ja asettelutiedostot sekä näkymän osat) käsittelemme tarkemmin hetken kuluttua,

mutta tässä mallissa huomionarvoista on se tapa, jolla useita erilaisia tapoja toimia on pyritty mahdollistamaan yhden "oikean" toimintatavan suosimisen sijaan.

Tämä ajatusmalli on hyvin linjassa myös itse ProcessWire-järjestelmän toimintatavan kanssa. ProcessWire on suunniteltu äärimmäisen joustavaksi järjestelmäksi, ja vaikka MVC-mallin tavoitteena onkin tuoda tiettyä määrämuotoisuutta sillä rakennettaviin toteutuksiin, emme kuitenkaan halua kahlita kehittäjää yhtään enempää kuin on tarpeen.

4.3.3 Käsittelijä

```
<?php

/**
 * Käsittelijä sivupohjalle 'home'
 *
 * Mikäli tällä sivupohjalla on tarve suorittaa omaa sovelluskoodia,
 * voidaan se tehdä tässä tiedostossa. Huom: sivupohjakohtainen
 * käsittelijä ei ole kuitenkaan pakollinen!
 *
 */

$view->placeholders->head = "<link rel='stylesheet' type='text/css'"
                          . " href='{$view->staticURL}css/home.css'>";
```

LISTAUS 21. MVC-mallin käsittelijä (/site/templates/controllers/home.php).

4.3.4 Näkymä

Puhumme tässä kappaleessa näkymästä, joskin todellisuudessa se on jaettu useampaan osaan. Näkymähakemiston alahakemistot ovat käytännössä näkymän alaryhmiä, joita ovat tässä tapauksessa layouts, partials ja scripts:

Layouts sisältää asettelutiedostot, eli kokonaiset sivupohjat, joiden sisään upotetaan näkymän sivukohtainen osuus. Useimmilla sivustoilla perusrakenne säilyy yhtenäisenä ja vain varsinaisen "sisältöosuus" rakenteineen – palstat ym. – vaihtelee sivuittain. Asettelutiedostot mahdollistavat juuri tämän. Asettelutiedostoja voisi olla useampiakin, mutta tässä esimerkissä hyödynnämme vain yhtä (default.php).

Partials sisältää sellaisia osia näkymästä, joita voidaan tarvita monessa näkymässä tai asettelutiedostossa – tai jopa useita kertoja saman näkymän tai asettelutiedoston sisällä. Esimerkkitoteutuksessamme täältä löytyy vain geneerinen ylävalikkomalli, mutta laajemmalla sivustolla tästä hakemistosta todennäköisesti löytyisi paljon muutakin.

Scripts pitää sisällään varsinaiset näkymätiedostot, jaoteltuina sivupohjan mukaan omiin hakemistoihinsa. Tässä ideana on se, että sama sivupohja voidaan tarjoilla eri asetteluilla, ja jopa eri tietotyypillä, riippuen tilanteesta ja tarpeesta. Esimerkkinä tästä home-sivupohjalle on lisätty vaihtoehtoinen näkymä, joka mahdollistaa kyseisen sivun sisällön esittämisen selaajalle JSON-muodossa.

```
<div id="body"><?= $page->body ?></div>
<div id="sidebar"><?= $page->sidebar ?></div>
```

LISTAUS 22. MVC-mallin basic-page -sivupohjan oletusnäkymä (/site/templates/views/scripts/basic-page/index.php).

```
<?php

/**
 * Asettelutiedosto, jossa paikat 'head'- ja 'content'-sisällöille
 *
 * Tämä tiedosto on tarkoitettu monikäyttöiseksi, eli moni sivupohja
 * voi hyödyntää samaa asettelutiedostoa ja sisällön osat välitetään
 * näkymältä asettelutiedostolle $placeholders-olion ominaisuuksina
 *
 */

?>
<html>
    <head>
        <title><?= $page->title ?></title>
        <link rel=stylesheet href="<?= $staticURL ?>css/style.css">
        <?= $placeholders->head ?>
    </head>
    <body>
        <?php include $partials->menu->top ?>
        <h1><?= $page->title ?></h1>
        <?= $placeholders->content ?>
    </body>
```

```
</html>
```

LISTAUS 23. Oletusasettelutiedosto, joka mahdollistaa saman perusrakenteen hyödyntämisen useille sivupohjille (/site/templates/views/layouts/default.php).

```
<ul id="menu-top">
    <?php
    $home = $pages->get('/');
    foreach ($home->and($home->children) as $item) {
        if (!$item->viewable()) continue;
        if ($item->id == $page->rootParent->id) {
            echo "<li class='active'>";
        } else {
            echo "<li>";
        }
        echo "<a href='{$item->url}'>{$item->title}</a></li>";
    }
    ?>
</ul>
```

LISTAUS 24. Yksittäinen näkymän osa eli "partial-tiedosto", joka vastaa sivuston ylävalikon piirtämisestä (/site/templates/views/partials/menu/top.php).

4.3.5 Staattiset resurssit

Yleensä web-sivustoon liittyy jonkin verran suoraan selaajalle näytettäviä "staattisia resursseja", eli kuva-, ääni-, tyyli-, ja JavaScript-tiedostoja, sekä kaikkea mahdollista muuta, joka ei sisällä palvelinpäässä suoritettavaa sovelluskoodia, ja joka ei siten myöskään liity suoraan aiemmin kuvattuihin MVC-mallin komponentteihin.

Staattisten resurssien hyödyntämiseen aihiotasolla ei liity mitään järjestelmäspesifiä, mutta esikäsittelijässä teimme kuitenkin erään pienen lisäyksen TemplateFile-olioihin: `$view->staticURL` palauttaa aina static-hakemiston julkisen web-osoitteen, jota puolestaan voidaan helposti hyödyntää näkymätiedostoissa. Kaikki alla näkyvät `<img>`-tagit tuottaisivat esimerkkisivustollamme täsmälleen saman lopputuloksen:

```
<img src="<?= $config->urls->templates ?>static/img/example.png"
     alt="" />
```

```
<img src="<?= $view->staticURL ?>img/example.png" alt="" />
<img src="/site/templates/static/img/example.png" alt="" />
```

LISTAUS 25. staattisen kuva-resurssin hyödyntäminen näkymätiedostossa.

Kaksi ensimmäistä viittaavat static-hakemiston sijaintiin hyödyntäen järjestelmän tietoa siitä, missä hakemistot tarkalleen sijaitsevat. Kolmas toimii useimmissa tapauksissa aivan yhtä hyvin, mutta tässä tapauksessa osoite on kovakoodattu, jolloin esimerkiksi sivuston siirtäminen toisen hakemiston alle, tai vaikkapa site-hakemiston nimeäminen uudelleen, aiheuttaisivat välittömästi ongelmia.

Static-hakemisto voidaan tarpeen mukaan jakaa pienempiin osiin, kuten tässä esimerkissä olemmekin jo tehneet – static/js/, static/css/, static/img/, jne. Tämä ei ole pakollista, eikä välttämättä aina käy järkeen, mutta mahdollisuus on kuitenkin olemassa; lopullinen hakemistorakenne riippuu pitkälti toteutettavan ratkaisun tarpeita, sekä kehittäjän omista preferensseistä.

Huomaathan myös, että static-hakemisto ei liity suoraan niihin tiedostoihin, jotka on tuotu tiedosto-kenttiin ProcessWire-järjestelmän keinoin. Näiden säilömisestä järjestelmä vastaa itse, ja ne löytyvät sivuston assets-hakemistosta (/site/assets/files/) aina sivun ID-tunnisteen mukaan kansioihin jaettuina.

4.4 Yhteenveto ja pohdinta

Tässä luvussa esitelty rakennemalli on käyttökelpoinen sellaisenaan, mutta luvun perimmäinen tavoite ei kuitenkaan ollut valmiiden ratkaisujen esittäminen. Huomattavasti tärkeämpää on ymmärtää, mihin ProcessWire alustana kykenee, ja mikä merkitys on sillä, ettei se pakota kehittäjän käyttöön esimerkiksi omaa variaatiotaan Model-View-Controller -mallista.

ProcessWiren vahvuus on mahdollistaminen – tapa, jolla se tekee lähes minkä tahansa ratkaisun toteuttamisen mahdolliseksi, tukien ja tehostaen kuitenkin samanaikaisesti kehittäjän työtä tarjoamalla pitkälle kehitettyjä toimintoja esimerkiksi sisällön hallintaan ja käyttäjien autentikointiin. ProcessWire ei ota kantaa siihen, millä tavalla kukin kehit-

täjä sitä päättää hyödyntää, ja juuri tästä syystä se tekee mahdolliseksi monenlaiset variaatiot myös rakennemallien osalta.

Esittelemämme MVC-malli on hyvin yksinkertainen, ja ProcessWiren päälle on toteutettu myös huomattavasti monipuolisempia variaatioita samasta teemasta. Osa näistä on asennettavissa moduuleina, osa toimitetaan sivustoprofiileina, mutta jos ProcessWire ja MVC yhdistelmänä vaikuttavat kiinnostavilta, kannattaa ehdottomasti alkuun tutustua valmiisiin vaihtoehtoihin.

Kuten monessa muussakin asiassa, oman ratkaisun toteuttaminen on usein oppimismielessä erittäin hyödyllistä, mutta parempaan lopputulokseen päästään yleensä hyödyntämällä valmiita ratkaisuja.

5 VÄLIMUISTIRATKAISUJEN HYÖDYNTÄMINEN

Yleisellä tasolla kuvattuna välimuisti (cache) on tapa tallentaa valmiiksi noudettua ja käsiteltyä tietoa sellaiseen muotoon, josta se on myöhemmin otettavissa käyttöön aiempaa helpommin, nopeammin ja tehokkaammin (Wessels 2001). Se, miten ja mihin tietoa tallennetaan – ja miten se sieltä noudetaan – riippuu välimuistin toteutustavasta.

Välimuistiratkaisujen kirjo on laaja, ja jonkinlainen välimuisti löytyykin käytännössä lähes jokaiselta kuviteltavissa olevalta tietojenkäsittelyn tasolta (Wessels 2001). Välimuisteja löytyy esimerkiksi fyysisistä komponenteista, käyttöjärjestelmistä, levyjärjestelmistä, käyttöjärjestelmien päällä pyörivistä apuohjelmista, tietokannoista, selaimista, ja jopa selaimissa ajettavista front-end -tason sovelluksista.

Tässä luvussa käsittelemme ensisijaisesti ProcessWiren tarjoamia välimuistiratkaisuja. Näillä viitataan sellaisiin toimintoihin, joiden avulla kehittäjä voi joko automaattisesti tai manuaalisesti ohjeistaa järjestelmää tallentamaan sivuja, sivujen osia, tai muuta sisältöä valmiiksi käsitellyssä muodossa levylle tai tietokantaan, josta se voidaan myöhemmin hakea ilman uuden käsittelyn vaatimaa työmäärää.

5.1 Välimuistiratkaisujen tarjoamat edut

Oikein hyödynnettynä välimuistin avulla voidaan lyhentää sovellusten suoritusaikoja, pienentää resurssien tarvetta, sekä saavuttaa useita yleisesti suorituskykyyn ja kustannustehokkuuteen liittyviä etuja. Web-ympäristössä välimuisti mahdollistaa myös sisällön tarjoamisen useiden eri palvelinten tai palvelinympäristöjen avulla tilanteesta riippuen, mistä hyvänä esimerkkinä toimivat erilaiset CDN-palvelut. (Wessels 2001).

Käytännön esimerkkinä välimuistin hyödyntämisestä ProcessWiren yhteydessä mainittakoon määrättyjen kenttien tulostaminen kaikilta sivuston sivuilta. Vaikka tällainen operaatio onkin yksinkertainen toteuttaa, sivumäärän kasvaessa se voi olla myös raskas suorittaa. Järkevämpää olisikin muodostaa näytettävä sisältö yhteen kertaan, tallentaa se valmiissa HTML-muodossa levylle tai tietokantaan, ja seuraavilla kyselyillä tulostaa suoraan kyseinen sisältö raskaan ja tarpeettoman nouto- ja käsittelyoperaation sijaan.

Levylle tallentaminen on tyypillinen, joskaan ei suinkaan ainoa, tapa tallentaa tietoa välimuistiin. Välimuistiratkaisu voi toimia myös tietokantatasolla, mihin palaamme tarkemmin hetken kuluttua. Lisäksi, kuten edellä ohimennen mainitsimme, välimuisti voidaan toteuttaa myös selaintasolla, jolloin palvelimen muistissa ei välttämättä säilytetä edes viittauksia välimuistitietueisiin, saati sitten itse välimuistiin tallennettuja tietoja.

Alla muutamia esimerkkejä niistä tilanteista, joissa välimuistin käyttö on erityisen tehokasta ja hyödyllistä – toisinaan jopa palvelun vakaan toiminnan kannalta pakollista:

- sivun tai sivunosan piirtäminen vaatii raskaita laskutoimituksia, paljon tekstin käsittelyä ja muotoilua, tai muulla tavoin erityisen monimutkaista logiikkaa,
- sivun tai sen osan piirtäminen vaatii tietokantayhteyksien muodostamista ja tietokantaan tallennetun tiedon noutamista,
- sivun tai sivunosan piirtäminen vaatii kommunikointia ulkopuolisten rajapintojen suuntaan, jolloin esimerkiksi verkkoviiveen merkitys korostuu, tai
- sivun tai sen osan piirtäminen vaatii suhteellisen suurten tietomäärien käsittelyä.

Paitsi paremmasta palvelusta, välimuistin hyödyntämisessä on kyse myös varautumisesta kunkin palvelun luonnolliseen kasvuun, sekä mahdollisiin hyökkäyksiin ja kiusantekoon. Mitä tehokkaampi sivusto on, sen suurempaa kuormaa se kestää, jolloin esimerkiksi viimeisten vuosien aikana yrityksille merkittävää haittaa aiheuttaneiden DDoS-tyyppisten hyökkäysten (Kaspersky Lab 2014) vaikutuksia pystytään lieventämään.

5.2 Erään välimuistiratkaisun hyödyllisyys lukuina

Oheinen taulukko auttaa hahmottamaan ProcessWiren sisäänrakennetun Template Cache -välimuistiratkaisun tuomia etuja mittaustulosten avulla. Tulokset perustuvat raskasta Skyscrapers-sivustoprofiilia suorittavan ProcessWire 2.3.0 -sivuston kuormittamiseen Apache Bench -testiohjelmalla sadan sivupyynnön ajan, kymmenen sivupyyntöä kerrallaan, perustason virtuaalipalvelinympäristössä.

Template Cache	Kokonaisaika	Minimi	Keskiarvo	Mediaani	Maksimi
Päällä	7.956 s	0.393 s	0.772 s	0.3545 s	3.472 s

Pois	32.445 s	1.05 s	3.148 s	3.178 s	5.68 s

TAULUKKO 1. Template Cache -välimuistiratkaisun vaikutus sivujen latausnopeuksiin. Luvut perustuvat Apache Bench -testiohjelman avulla saatuihin tuloksiin.

Kuten oheisesta taulukosta on nähtävissä, välimuistin hyödyntäminen voi parantaa sivuston latausnopeuksia moninkertaisesti. Hyödyn mitattavissa oleva määrä vaihtelee tapauskohtaisesti, mutta välimuistista on kuitenkin lähes aina jonkin verran hyötyä.

Välimuistin koostaminen vaatii hieman ylimääräistä työtä, ja yleensä sisällön muuttuessa välimuisti joudutaan kokonaan tai osittain tyhjentämään (alustamaan), mutta useimmissa verkkopalveluissa sisältöön kohdistuvia lukuoperaatioita tapahtuu kuitenkin vähemmän kuin kirjoitusoperaatioita. Tällöin luonnollisesti lukuoperaatioiden optimointiin kannattaakin keskittää enemmän resursseja.

Mitä raskaammasta sivustosta tai sovelluksesta on kyse, sen suurempi etu välimuistin hyödyntämisestä saadaan. Lisäksi välimuistin hyödyllisyyteen vaikuttaa erittäin oleellisesti osumatodennäköisyys: kuinka suureen osaan sovellukseen kohdistuvista kyselyistä voidaan vastata suoraan välimuistista. (Coggeshall 2009: 83-85).

5.3 ProcessWiren tarjoamat välimuistiratkaisut

ProcessWire tarjoaa neljä sisäänrakennettua välimuistiratkaisua: Template Cache, Markup Cache, Fieldtype Cache, ja WireCache. Lisäksi järjestelmän tapa tallentaa kuvista tehtävät koko- ym. variaatiot tiedostoina levylle toimii eräänlaisena välimuistiratkaisuna, tietyt järjestelmän sisäiset tiedot säilytetään automaattisesti välimuistissa (vrt. moduulivälimuisti), ja saatavilla on myös kaupallinen ProCache-välimuistimoduuli.

Alla kaikki edellä mainitut ProcessWiren välimuistiratkaisut lyhyesti esiteltyinä ja tarpeen mukaan esimerkkikoodilla täydennettyinä. Ratkaisuja on näin monta siitä yksinkertaisesta syystä, että kullakin niistä on omat hyvät ja huonot puolensa, eikä mikään niistä ole kaikin tavoin – saati kaikissa tilanteissa – yksiselitteisesti muita parempi.

5.3.1 Template Cache

Template Cache sopii useimmille sivustoille, ja onkin ensisijainen ja suositelluin ratkaisu sekä helppokäyttöisyytensä että tehokkuutensa vuoksi. Kun Template Cache on otettu käyttöön, se on käytännössä täysin automaattinen ratkaisu.

Tämä järjestelmän sisäänrakennettu toiminnallisuus otetaan käyttöön sivupohjan asetuksista, määrittämällä kuinka pitkäksi aikaa kyseistä sivupohjaa hyödyntävät sivut tallennetaan välimuistiin. Muut Template Cachen toimintaan vaikuttavat asetukset liittyvät tilanteisiin, joissa välimuisti tyhjennetään, sekä siihen, hyödynnetäänkö välimuistia myös tunnistautuneiden käyttäjien kohdalla.

Helppokäyttöisyydestään huolimatta – tai juuri siitä johtuen – Template Cacheen liittyy kuitenkin muutamia olennaisia rajoituksia, jotka vaikuttavat sen käyttöön, tosin lähinnä hieman monimutkaisemmissa tilanteissa:

- Template Cache ei huomioi GET- tai POST-parametreja. Sivupohjan asetuksissa on kuitenkin mahdollista nimetä ne parametrit, joiden osalta välimuisti tulisi ohittaa kokonaan. Lisäksi on mahdollista käyttää tähti-notaatiota (*), jolloin kaikki kyseisen tyypin parametrit ohittavat välimuistin.
- Template Cache ei tallenna URL-segmenttien sisältöä välimuistiin. Mikäli URL-segmentit on otettu sivupohjan asetuksista käyttöön, niiden osalta välimuisti ohitetaan automaattisesti.
- Template Cache tallentaa kokonaisia sivuja sellaisenaan, eikä sen avulla välimuistiin tallennettuun sisältöön ole helposti sisällytettävissä esimerkiksi palvelinpäässä luotavaa dynaamista sisältöä.

5.3.2 Markup Cache

Siinä missä Template Cache on täysin automaattinen välimuistiratkaisu, Markup Cache puolestaan on manuaalinen työkalu, jonka avulla sivupohjien aihiotiedostoissa voidaan ohjelmallisesti ja tapauskohtaisesti tallentaa merkkijonoja (esimerkiksi HTML-, JSON- tai XML-tietueita) välimuistiin.

Markup Cachen tyypillisimmät käyttökohteet ovat niitä, joissa kokonaisen sivun tallentaminen välimuistiin ei, esimerkiksi erityisen dynaamisen sisällön vuoksi, ole mahdollista. Tehokkuudessa Markup Cache kuitenkin häviää Template Cachelle, joten sen käyttöä suositellaan vain kun sille on oikeaa tarvetta, joka ei ole Template Cachen keinoin ratkottavissa.

Koska sisältö sekä tallennetaan, että haetaan välimuistista ohjelmallisesti, sen muokkaus PHP-kielen tarjoamin toiminnoin sekä ennen tallennusta, että tallennuksen jälkeen tekee tästä ratkaisusta erityisen joustavan. Markup Cachen hyödyntäminen aihiotiedostossa näyttää käytännössä seuraavalta:

```
// alustetaan Markup Cache -olio
$cache = $modules->MarkupCache;

// noudetaan välimuistista sisältö tunnisteella "breadcrumbs";
// jälkimmäinen parametri määrittää, kuinka pitkäksi aikaa
// sisältö välimuistiin tallennetaan
if (!$data = $cache->get("breadcrumbs", 3600)) {

    // välimuistista ei vielä löytynyt sisältöä tunnisteella
    // "breadcrumbs", joten uusi sisältö nyt ja tallennetaan
    // se välimuistiin myöhempää käyttöä varten
    $data = "<div id='breadcrumbs'>";
    foreach ($page->parents as $parent) {
        $data .= "<a href='{$parent->url}'>{$parent->title}</a> › ";
    }
    $data .= "{$page->title}</div>";

    // tallennetaan $data välimuistiin nimellä "breadcrumbs"
    // (Markup Cache muistaa nimen, joka aiemmin syötettiin
    // $cache->get() -metodille)
    $cache->save($data);

}

// tulostetaan merkkaus: tässä kohtaa sillä ei ole merkitystä,
// löytyikö merkkaus välimuistista vai luotiinko se lennossa
echo $data;
```

LISTAUS 26. Esimerkki Markup Cache -välimuistiratkaisun hyödyntämisestä aihiotasolla.

Markup Cache on nimensä mukaisesti ensisijaisesti suunniteltu merkkauksen tallentamiseen välimuistiin, mutta soveltuu käytännössä minkä tahansa tekstimuodossa esitettävissä olevan sisällön tallentamiseen. Seuraavaksi käsittelemme WireCache-ratkaisua, joka on ProcessWiren version 2.5 mukana tullut toiminnallisuus, ja jota nykyisin suositellaan käytettäväksi Markup Cachen sijaan uusissa toteutuksissa.

5.3.3 WireCache

WireCache on tässä luvussa käsitellyistä työvälineistä uusin, ja toimii pitkälti kuten Markup Cache, jonka seuraajaksi se järjestelmään lisättiinkin. Suurimmat erot mainittujen ratkaisujen välillä ovat siinä, miten niiden tallentamiin tietoihin päästään käsiksi, sekä siinä, mihin tieto tarkalleen tallennetaan:

- Markup Cache alustetaan käyttöön noutamalla uusi MarkupCache-luokan instanssi (kts. aiemmat esimerkit), kun taas WireCache näkyy aina kehittäjien saatavilla olevana API-muuttujana nimeltä $cache.
- Markup Cache tallentaa tietonsa levylle tiedostoina, WireCache puolestaan tallentaa ne tietokantaan.

WireCachen edut Markup Cacheen verrattuna perustuvat sen kehittäjärajapinnan helppokäyttöisyyteen, laajoihin jatkokehitysmahdollisuuksiin, tietokantapohjaisen välimuistin toimivuuteen vaihtelevissa ympäristöissä – joissa esimerkiksi levylle kirjoittaminen ei aina ole aivan yksinkertaista – sekä muihin tietokannan tuomiin uusiin mahdollisuuksiin, kuten esimerkiksi parempaan tietoturvaan, replikointiin, sekä monipuolisiin hakutoimintoihin.

Oheinen esimerkki on sama, jota aiemmin käytimme Markup Cachen esittelyyn, mutta sovitettuna WireCachen kehittäjärajapinnalle:

```
// noudetaan välimuistista sisältö tunnisteella "breadcrumbs"
if (!$data = $cache->get("breadcrumbs")) {

    // välimuistista ei vielä löytynyt sisältöä tunnisteella
    // "breadcrumbs", joten uusi sisältö nyt ja tallennetaan
    // se välimuistiin myöhempää käyttöä varten
```

```
    $data = "<div id='breadcrumbs'>";
    foreach ($page->parents as $parent) {
        $data .= "<a href='{$parent->url}'>{$parent->title}</a>"
               . " › ";
    }
    $data .= "{$page->title}</div>";

    // tallennetaan $data välimuistiin nimellä "breadcrumbs";
    // kolmas parametri määrittää, kuinka pitkäksi aikaa
    // sisältö välimuistiin tallennetaan
    $cache->save("breadcrumbs", $data, 3600);

}

// tulostetaan merkkaus: tässä kohtaa sillä ei ole merkitystä,
// löytyikö merkkaus välimuistista vai luotiinko se lennossa
echo $data;
```

LISTAUS 27. Esimerkki WireCache -välimuistiratkaisun hyödyntämisestä aihiotasolla.

Yksinkertaisten merkkijonojen lisäksi WireCache kykenee tallentamaan välimuistiin myös array-muotoista sisältöä (automaattisesti JSON-muotoon käännettynä tallennusta varten), ja jatkosuunnitelmissa on myös ProcessWire-järjestelmän PageArray-olioiden automaattinen tallennus välimuistiin.

5.3.4 Fieldtype Cache

Fieldtype Cache tallentaa WireCachen tapaan välimuistiin lisättävät tiedot tietokantaan. Kyseessä on kenttätyyppi, jota hyödyntävien kenttien asetuksista voidaan määrittää muita kenttiä seurattaviksi. Jos ja kun seurattaviin kenttiin tulee muutoksia, niiden tiedot yhdistetään automaattisesti yhdeksi JSON-muotoiseksi tietueeksi tietokantaan.

ProcessWiren tietokantarakenteessa kullakin kentällä on omat taulunsa, joista tarvittavat tiedot kootaan sivuja noudettaessa JOIN-lausekkeiden avulla yhteen. Tämä on skaalautuvuuden kannalta joustava ratkaisu, koska sivun kentistä voidaan käytännössä noutaa vain ne, joita kyseisessä tilanteessa tarvitaan. Luonnollisesti se, montako kenttää kulloinkin tietokannasta haetaan, vaikuttaa myös kyselyiden raskauteen.

Tämä korostuu tilanteessa, jossa sivuja haetaan suurella määrällä kriteerejä:

```
$john_does = $pages->find('field1|field2|field3|...|field15%="john doe"');
```

LISTAUS 28. Esimerkki sivujen etsimisestä, kun hakusana voi sijaita missä tahansa useista erillisistä hakukentistä.

Fieldtype Cache mahdollistaa edellä mainitun kyselyn merkittävän tehostamisen, esimerkiksi näin:

```
$john_does = $pages->find('my_cache_field%="john doe"');
```

LISTAUS 29. Esimerkki sivujen etsimisestä, kun hakukenttien sisältö on yhdistetty yhteen Fieldtype Cache -kenttään.

Koska Fieldtype Cache tallentaa kenttien sisällön JSON-muotoon, se mahdollistaa käytännössä hieman monimutkaisemmatkin käyttötapaukset. Esimerkissä yllä haetaan vain yksittäistä merkkijonoa välimuistiin tallennetusta JSON-muotoisesta sisällöstä, mutta kentän JSON-sisältöä voidaan myös hyödyntää sivustolla alkuperäisessä muodossaan:

```
$cached_fields = json_decode($page->my_cache_field);
echo "Field 1: " . $cached_fields['field1'] . "<br />";
echo "Field 2: " . $cached_fields['field2'] . "<br />";
echo "Field 3: " . $cached_fields['field3'] . "<br />";
// jne.
```

LISTAUS 30. Fieldtype Cache -kenttään tallennettujen arvojen tulostaminen.

5.3.5 Kaupallinen ProCache-moduuli

Tähän mennessä esitellyt välimuistiratkaisut ovat olleet järjestelmän sisäänrakennettuja toimintoja. ProCache eroaa joukosta varsin merkittävästi, sillä kyse on kaupallisesta kolmannen osapuolen ratkaisusta, joka asennetaan järjestelmään erillisenä moduulina. Syy ProCachen esittelyyn tässä yhteydessä onkin paitsi sen muista välimuistiratkaisuista poikkeava toimintatapa, myös sen varsin erikoislaatuinen taustatarina.

Ensinnäkin ProCache-moduulin taustalta löytyy ProcessWire-järjestelmän pääkehittäjä Ryan Cramer (Cramer 2015a). Kyse on siis kaupallisesta kolmannen osapuolen moduulista, mutta moduulin taustalta löytyvä taho on kuitenkin sama kuin järjestelmällä itsellään. Toinen hyvin merkittävä eroavaisuus löytyy siitä, miten moduuli teknisesti toimii, sillä sen ideana on ohittaa sivukyselyihin vastattaessa paitsi ProcessWire, myös itse PHP-kieli.

Template Cachen tapaan ProCache tallentaa sivun sisällön renderöintivaiheessa levylle, mutta tähän loppuvatkin työkalujen yhtäläisyydet. Siinä missä Template Cache vaatii kunkin kyselyn kohdalla ProcessWire-järjestelmän käynnistämistä, ProCache tarjoilee sisällön .htaccess-sääntöjen avulla. Käytännössä lopputulos ja saavutettu tehokkuus ovat samaa luokkaa kuin tarjoiltaessa staattisia HTML-tiedostoja suoraan levyltä.

1. Ei välimuistia	2. Template Cache	3. ProCache
29 s	6 s	0.017 s

TAULUKKO 2. Vertailutietoa erittäin raskaan sivun latausnopeuksista, kun välimuisti ei ole käytössä, hyödynnetään sisäänrakennettua Template Cache -välimuistiratkaisua, ja kun hyödynnetään ProCache-välimuistimoduulia. Tässä esitettyjen tietojen lähteenä on ProCache-moduulin markkinointimateriaali.

PHP:n sivuuttamisesta johtuen dynaamisten palvelinpään ominaisuuksien yhdistäminen ProCachen toimintamalliin on hankalaa. Mikäli kunkin sivukyselyn on tarpeen laukaista jokin palvelinpään toiminto, tämä on ProCachen yhteydessä usein yksinkertaisinta toteuttaa AJAX-menetelmällä. ProCachen suorituskyky on kuitenkin siinä määrin omaa luokkaansa, että useimmiten tämä painaa vaa'assa enemmän kuin dynaamisten palvelinpään toimintojen hankaloituminen.

Koska ProCache on suunnattu kokonaisvaltaiseen sivuston optimointiin, sen ominaisuudet eivät rajoitu välimuistin hallintaan. Tuki CDN-integraatiolle mahdollistaa sivuston binäärien (mm. kuvat ja tiedostot) automaattisen viennin ulkoiseen CDN-palveluun, CSS- ja JavaScript-tiedostot voidaan ProCachen avulla automaattisesti yhdistää, HTML-merkkauksen minimointiin löytyy oma toimintonsa, ja niin edelleen.

5.3.6 Muut välimuistiratkaisut lyhyesti

ProcessWire tallentaa automaattisesti välimuistikopioita kaikista järjestelmän työkaluin muokattavista kuvista. Kuvien käsittely on suhteellisen raskas toimenpide, ja tällä tavoin varmistetaan, että esimerkiksi kertaalleen tiettyyn kokoon skaalatusta kuvasta säilytetään sekä alkuperäinen versio, että uusi, valmiiksi haluttuun kokoon skaalattu versio.

Toinen lähes täysin automatisoitu välimuisti on moduulivälimuisti, eli listaus kulloinkin asennetuista moduuleista tietoineen. Tämä ei ole kehittäjän kannalta yleensä erityisen merkittävä välimuistin muoto, mutta esimerkiksi uuden moduulin lisäämisen jälkeen moduulivälimuisti on yleensä erikseen tyhjennettävä.

Järjestelmä tallentaa myös yksittäisen kyselyn aikana haetut moduulit, sivut, ja muut tiedot välimuistiin, jotta turhilta tietokantakyselyiltä voidaan välttyä. Kyselymoottori (selector engine) toimii tästä syystä kahdella tasolla: ensimmäinen kysely toimii yleensä tietokantatasolla (database selectors), kun taas jo aiemmin muistiin haettujen sivujen tarkempi rajaus tapahtuu muistissa (in-memory selectors).

Vaikka tässä esitellyt välimuistiratkaisut ovatkin pitkälti automaattisia, eivätkä siten vaikuta oleellisesti kehittäjän työskentelyyn, niiden olemassaolosta on kuitenkin hyvä olla tietoinen. Moduulivälimuistin tyhjentäminen on hyvä esimerkki tilanteesta, jossa tällä tiedolla on merkitystä. Vastaavasti kyselymoottori toimii hieman eri riippuen siitä, ajetaanko tietty kysely tietokantatasolla vai muistissa.

5.4 Yhteenveto ja pohdinta

Välimuistiratkaisujen hyödyntäminen auttaa web-sivustoja ja -palvelimia kahden äärimmäisen tärkeän tavoitteen, nopeuden ja toimintavarmuuden, saavuttamisessa. Tehokkaan välimuistin hyödyntämisen ansiosta palvelin pystyy palvelemaan suurempia käyttäjämääriä nopeammin ja pienemmillä resursseilla, jolloin myös todennäköisyys palvelun estymiseen yllättävän ylikuormitustilanteen sattuessa on pienempi.

ProcessWire tarjoaa välimuistiratkaisujen saralla kattavat työkalut aina yksittäisten sivunosien tai merkkijonojen manuaalisesta käsittelystä kokonaisten sivujen ja sivustojen

täysin automaattisiin välimuistiratkaisuihin. Kun mukaan lasketaan vielä esimerkiksi ProCachen kaltaiset kolmannen osapuolen välimuistiratkaisut, jotakuinkin jokaiseen kuviteltavissa olevaan tilanteeseen on olemassa oma ratkaisunsa.

Aiemmin tässä luvussa esiteltiin lukuja, jotka puhuivat selkeää kieltä tehokkaan välimuistin hyödyntämisen tuomista eduista: kunkin sivupyynnön palvelemiseen voidaan helposti käyttää useita, ellei jopa useita kymmeniä – tai satoja – kertoja vähemmän aikaa ja resursseja. Tämän luulisi jo riittävän perusteluksi sille, miksi välimuistin käyttö on useimmissa tilanteissa erittäin suositeltavaa.

Koska ratkaisuja on useaan lähtöön, valistuneen kehittäjän on kuitenkin hyvä olla selvillä niiden hyödyistä ja haitoista, jotta hän voi tehdä järkeviä päätöksiä asian suhteen. Joustavalle ja mukautuvalle luonteelleen ominaisella tavalla ProcessWire ei pakota kehittäjää hyödyntämään ainuttakaan tässä mainituista työvälineistä, sillä päätösvallan tässäkin asiassa tulisi loppukädessä aina olla kehittäjällä itsellään.

6 MODUULIKEHITYKSEN PERUSTEET

Suurin osa ProcessWiren parissa työskentelevän kehittäjän ajasta kuluu todennäköisesti sivuaihioiden ja hallintanäkymien parissa, ja näiden avulla on mahdollista toteuttaa monipuolisia ja laajoja ratkaisuja. Moduulit kuitenkin mahdollistavat tiettyjä asioita, joita aihio- tai käyttöliittymätasoilla ei joko ole mahdollista toteuttaa, tai joiden toteuttaminen tällä tasolla on tehotonta – tai aiheuttaa kohtuutonta toistoa.

Tässä luvussa perehdymme moduulien kehittämisen perusteisiin, käymme uuden moduulin kehittämiseen liittyvät työvaiheet kohta kohdalta läpi, ja lopulta toteutamme hyvin yksinkertaisen esimerkkimoduulin. Aivan ensimmäiseksi pureudumme kuitenkin siihen, mitä moduulit oikeastaan ovat, ja mihin niitä tarvitaan.

6.1 Mihin moduuleja tarvitaan?

ProcessWiren avulla voidaan rakentaa kokonaisia sivustoja ja sovelluksia toteuttamatta ainuttakaan omaa moduulia, mutta moduulit voivat helpottaa monimutkaisia tehtäviä, niiden avulla voidaan suorittaa toistuvia tehtäviä tehokkaasti ja DRY-periaatetta noudattaen, ja ne myös mahdollistavat joitakin sellaisia tehtäviä, jotka muutoin olisivat mahdottomia.

Markup-moduulit ovat yksi esimerkki siitä, miten toistuvia tehtäviä on mahdollista tehostaa. Markup-moduuleja voidaan kutsua esimerkiksi aihiotiedostoissa, ja niiden tehtävänä on tuottaa määrämuotoista merkkausta annettujen parametrien perusteella. MarkupRSS muuttaa PageArrayn valmiiksi RSS-syötteeksi, MarkupSimpleNavigation tulostaa HTML-valikkorakenteen yhdellä komentokutsulla, ja niin edelleen.

Toinen hyvä esimerkki ovat Textformatter-moduulit, joita useimmille tekstiä sisältäville kentille voidaan ottaa käyttöön yksi tai useampia. Kun kentän arvoa kysytään esimerkiksi aihiotiedostossa, arvo toimitetaankin ensin käyttöön otetuille Textformatter-moduuleille yksi kerrallaan; ne muokkaavat arvoa sääntöjensä mukaisesti ja lopulta palauttavat muokatun arvon joko kysyjälle tai seuraavalle Textformatter-moduulille.

Aihiotasolla voidaan esimerkiksi hakea ja hyödyntää sivuja ja niiden sisältöä, sekä käsitellä käyttäjän antamaa syötettä. Tällä tasolla ei ole kuitenkaan mahdollista määritellä uutta kenttätyyppiä, toteuttaa uutta näkymää sivuston ylläpitotilaan, saati automatisoida sivun tallennuksen yhteydessä tapahtuvia toimenpiteitä. Nämä ovat esimerkkejä asioista, jotka ovat mahdollistettavissa juuri moduulien avulla.

6.2 Mitä moduulit oikeastaan ovat?

ProcessWire-moduulit ovat tiettyjen sääntöjen mukaan laadittuja luokkia, joista jokainen sijaitsee omassa .module-päätteisessä tiedostossaan sivuston moduulihakemistossa. Moduulihakemistoja on kaksi, /site/modules/ sivustokohtaisille moduuleille ja/wire/modules/ järjestelmän mukana toimitettaville moduuleille, mutta kehittäjän ei tyypillisesti ole tarpeen muokata mitään /wire/ -hakemiston alta löytyvää.

Kaikki moduulihakemistoista löytyvät moduulit eivät välttämättä ole samanaikaisesti käytettävissä, sillä ennen käyttöä moduuli tulee ensin asentaa. Asentaminen onnistuu joko järjestelmän graafisen hallintakäyttöliittymän avulla, tai ohjelmallisesti $modules-muuttujan avulla: `$modules->install('MarkupSimpleNavigation')`. Asennuksen jälkeen moduuli on yleensä heti käytettävissä, joskin osa moduuleista vaatii konfigurointia.

Asennetut moduulit voivat olla autoload-tyyppisiä, jolloin ne ladataan automaattisesti järjestelmän mukana, tai ne voivat vaatia erillistä kutsua, esimerkiksi tähän tapaan: `$modules->get('MarkupSimpleNavigation')`. Tällä on varsin olennainen merkitys järjestelmän toiminnan kannalta, sillä vaikka autoload-toiminto onkin esimerkiksi koukkujen hyödyntämisen kannalta tarpeellinen, jokainen ladattu moduuli kuluttaa resursseja, eikä moduuleja näin ollen kannata asentaa tai ladata turhaan.

6.2.1 Moduulirajapinnat

Olio-ohjelmoinnissa rajapinnat (interfaces) määrittävät ne metodit ja ominaisuudet, jotka kyseisen rajapinnan toteuttavan luokan tulee sisältää (Zandstra 2008: 51), eli ne voidaan nähdä eräänlaisina sopimuksina. ProcessWiren tapauksessa moduulirajapintoja hyödynnetään kuitenkin lähinnä moduulien tunnistamiseen, ja niissä määritellään vain minimaalinen määrä pakollisia metodeja.

Kukin moduuli noudattaa ProcessWire tarjoamaa moduulirajapintaa Module, minkä lisäksi konfigurointiasetuksia sisältävät moduulit toteuttavat myös rajapinnan ConfigurableModule. Käytettävät rajapinnat voidaan määrittää implements-avainsanan avulla, ja niitä voi tarvittaessa olla useampiakin: class ModuleName implements Module, ConfigurableModule, MyOwnInterface, ...

6.2.2 Kantaluokat ja periytyminen

Myös periytyminen on olio-ohjelmoinnin käsite, ja tarkoittaa käytännössä sitä, että esimerkiksi WireData-luokan perivä moduuliluokka sisältää kaikki ne kantaluokan metodit ja ominaisuudet, joita ei ole erikseen määritelty yksityisiksi näkyvyysmääreellä private (Zandstra 2008: 35-36). Periytyminen määritellään extends-avainsanan avulla: class ModuleName extends WireData implements Module.

Uusi moduuli perii tyypillisesti jonkin järjestelmän varsinaisista kantaluokista tai valmiista moduuleista. Valmiin moduulin periminen mahdollistaa kyseisen moduulin toimintojen laajentamisen tai korvaamisen, eli vanhan moduulin hyödyntämisen pohjana uudelle, siinä missä tietyn kantaluokan periminen mahdollistaa jokseenkin tyhjältä pöydältä aloittamisen.

ProcessWiren varsinaiset kantaluokat kuuluvat ydinkoodiin, ja kullakin niistä on omat ominaispiirteensä ja merkityksensä. Valmiita kantaluokkia ovat esimerkiksi Process, Fieldtype, Inputfield, ja WireMail – sekä edellä mainittu WireData, joka edustaa eräänlaista ylätason sisältövarastoa, ja toimii kantaluokkana suurelle osalle kaikista muista ProcessWiren luokista.

Siinä missä yksi luokka voi toteuttaa useita rajapintoja, PHP ei tue moniperintää. Tämä tarkoittaa sitä, että jokainen luokka voi periytyä vain yhdestä kantaluokasta. Periytyminen toimii kuitenkin hierarkkisesti, eli moduuliluokka voi periytyä toisesta moduuliluokasta, joka puolestaan periytyy WireData-luokasta, ja niin edelleen. Tällöin moduuli perii paitsi kantaluokkansa, myös sen kantaluokan ominaisuudet.

Pohjaluokista esimerkiksi Process, Fieldtype, Inputfield, ja WireMail toimivat kantaluokkina tiettyyn tehtävään erikoistuneille moduuleille, jolloin puhutaan myös *moduulityypeistä*. WireData puolestaan soveltuu kantaluokaksi niille moduuleille, joiden tarkoi-

tuksena on koukkujen hyödyntäminen, tai joilla on jokin muu sellainen tehtävä, joka ei suoraan liity mihinkään valmiista moduulityypeistä.

6.2.3 Moduulityypit

Moduulin tyyppi määrittää sen tarkoituksen järjestelmän sisällä, sekä ainakin osan siltä vaadituista ominaisuuksista. Process-moduulit (ylläpitoprosessit) edustavat sivuston hallinnassa yksittäisiä näkymiä, Fieldtype-moduulit (kenttätyypit) määrittävät kunkin kentän tietorakenteen, ja Inputfield-moduulit (lomakekentät) puolestaan toimivat ensisijaisesti kenttien näkyvinä ilmentyminä ylläpitonäkymissä.

Moduulin tyypin määrää yleensä kantaluokka, mutta poikkeuksia tähän sääntöön ovat esimerkiksi Markup-moduulit, sillä järjestelmä ei tunne erillistä Markup-kantaluokkaa. Kyseisen moduulityypin edustajat tunnistaakin tarkoituksen, ei niinkään toteutustavan, perusteella. Tyypittömillä moduuleilla puolestaan viitataan moduuleihin, joiden suorittamat tehtävät eivät kuulu minkään yksittäisen moduulityypin tehtävänkuvaan.

Moduulityyppi	Pohjaluokka	Kuvaus
Process	Process	Process-moduulit edustavat näkymiä sivuston hallinnassa. Process-moduulit toimivat pitkälti kuten sivut, ja myös niillä on omat URL-osoitteensa, sekä tarvittaessa useita ala-osoitteita (vrt. sivujen osalta URL-segmentit).
Fieldtype	Fieldtype, FieldtypeMulti	Fieldtype-moduulit eli kenttätyypit edustavat kenttien tietorakenteita ja tietokantaskeemoja, sekä vastaavat arvojen sanitoinnista. Kentällä voi olla useita virtuaalisia tietueita, eikä tietokantaskeema aina vastaa suoraan kentän rajapintaa.
Inputfield	Inputfield	Inputfield-moduulit edustavat lomakekenttiä, ja ovat siten kenttätyyppien näkyviä

		ilmentymiä esimerkiksi sivuston hallinta-työkaluissa ja mahdollisissa lomakenäky-missä.
Textformatter	Textformatter	Textformatter-moduulit vaikuttavat kentän sisällön muotoiluun tulostusvaiheessa. Kullakin kentällä voi olla samanaikaisesti käytössään useita Textformatter-moduuleja, jolloin ne toimivat kukin vuo-rollaan, kentän asetusten määräämässä järjestyksessä.
Markup	WireData	Markup-moduulien tehtävänä on tuottaa valmista, usein HTML-muotoista, merk-kausta niille annettujen parametrien poh-jalta. Markup-moduuleilla ei ole omaa pohjaluokkaa, ja myös niiden toteutustapa, rajapinta, ja käyttötarkoitus voivat vaihdel-la merkittävästi.
File Validator	FileValidatorModule	File Validator -moduulien avulla voidaan varmistaa sivustolle tuotavien tiedostojen oikeellisuus. Väärin muotoiltu tiedosto voidaan joko sanitoida, tai suoraan hylätä.
Admin Theme	AdminTheme	Admin Theme -moduulit määrittävät ylläpitonäkymän ulkoasun, asettelun, ja tietyt toiminnot. Mikäli useita Admin Theme -moduuleja on asennettu samanaikaisesti, kulloinkin käytössä oleva teema voidaan valita käyttäjäkohtaisesti.
JS, Jquery	ModuleJS	JS-moduulit (myös "Jquery-moduulit") on tarkoitettu tietyn JavaScript-kirjaston tai muun paketoidun kokonaisuuden sisällyt-tämiseen ylläpitonäkymiin, julkiselle si-

		vustolle, tai molempiin.
Markup Fieldtype	MarkupFieldtype	Markup Fieldtype -moduulit mahdollistavat kenttien arvojen tulostamisen valmiiseen HTML-muotoon kenttätyypin ominaisuuksista riippuen.
Action	WireAction, PageAction	Action-moduulit ovat tapa paketoida monikäyttöiseen muotoon tehtäviä, joita ulkopuolinen moduuli voi tarpeen mukaan käynnistää. Tehtävä voidaan aina kohdistaa yhteen tai useampaan sivuun tai muuhun järjestelmän olioon.
WireMail	WireMail	WireMail-moduulit toteuttavat WireMailInterface-rajapinnan ja mahdollistavat sähköpostin lähettämisen ProcessWiren sisäänrakennettuja toimintoja hyödyntäen myös muilla tavoin kuin oletuksena aina käytettävän PHP:n mail()-funktion avulla.
Comment Filter	CommentFilter	Comment Filter -moduulit on suunniteltu kommenttien suodatukseen kommentointimoduulin yhteydessä.

TAULUKKO 3. moduulityypit kuvauksineen ja kantaluokkineen.

6.2.4 Moduulien nimeäminen

Mitä moduulien nimeämiseen tulee, moduuliluokan nimi tulisi aina aloittaa sen tyypillä: ProcessPageEdit, MarkupSimpleNavigation, InputfieldText, TextformatterEntities, WireMailSwiftMailer, ja niin edelleen. Lisäksi ydinkomponenttiin tai toiseen moduuliin ominaisuuksia lisäävän moduulin pitäisi hyödyntää alkuperäisen komponentin tai moduulin nimeä osana omaa nimeään: PageRender, PagePermissions, ja niin edelleen.

Moduulien nimeämisessä on usein kyse lähinnä sovituista käytännöistä, mutta moduuliluokan nimeä voidaan myös käyttää nopeana ja tehokkaana tapana löytää kaikki sopivat moduulit. `$modules->find("className^=Textformatter")` esimerkiksi löytää kaikki ne moduulit, joiden luokan nimi alkaa sanalla "Textformatter"; väärin nimetty moduuli voi jäädä huomioimatta, eikä siten toimikaan kaikissa tilanteissa oikein.

Huomaathan, että moduulin nimellä viitataan tässä yhteydessä moduuliluokan nimeen, ei siis moduulin selkokieliseen otsikkoon (title-tietueeseen). Monissa tapauksissa moduulin otsikko on jopa monikielinen tietue, jolloin kullakin sivustolla käytössä olevalla kielellä voi olla siitä oma versionsa!

6.3 Oman moduulin rakentaminen vaiheittain

Tässä osiossa käymme tiiviisti läpi tärkeimmät oman moduulin rakentamiseen liittyvät työvaiheet. Lopuksi esittelemme erittäin yksinkertaisen Textformatter-moduulin, joka on kuitenkin hyvä lähtökohta omien, monimutkaisempien moduulien toteuttamiseen.

6.3.1 Esivalmistelut

Kenties yksinkertaisin tapa aloittaa oman moduulin parissa työskentely on jonkin valmiin moduulin valitseminen pohjaksi. Erityisesti tätä tarkoitusta varten löytyy myös valmiita moduulialkioita: HelloWorld.module, joka esittelee ProcessWiren moduulien toimintaa ja erityisesti koukkujen hyödyntämistä, sekä ProcessHello.module, joka toimii pohjana järjestelmän hallintaliittymään toteutettaville uusille näkymille.

Vaihtoehto moduulialkioiden tai valmiiden moduulien hyödyntämiselle uuden moduulin lähtökohtana on web-pohjainen ProcessWire Module Generator, eli modules.pw, jossa uuden moduulin perustiedot syötetään lomakkeen kenttiin, minkä jälkeen työkalu tarjoaa ladattavaksi määritysten mukaisen moduulirungon (Knoll 2015). ProcessWire Module Generator on integroitu myös wireshell-työkaluun, mikä mahdollistaa moduulirungon luonnin suoraan komentokehotteen kautta (Herrmann 2015).

Moduulien ja moduulitiedostojen nimeämiseen on olemassa käytännöt, joita tulee noudattaa, jotta järjestelmä tunnistaa moduulin oikein. Ensinnäkin moduulin tiedoston tulee noudattaa sen nimeä: mikäli moduulin nimi on HelloWorld, moduulitiedoston nimen

tulee olla HelloWorld.module, HelloWorld.module.php, tai HelloWorldModule.php. Lisäksi moduulin nimi määrittää myös moduuliluokan nimen: class HelloWorld extends WireData implements Module.

Kun moduulia lopulta lähdetään muokkaamaan, se on suositeltavaa ensin asentaa ei-julkiselle sivustolle, ja lisäksi kyseiselle sivustolle kannattaa ottaa käyttöön debug-tila asettamalla sivuston config.php-asetustiedostossa muuttujan $config->debug arvoksi true. Debug-tila tuo kehittäjän kaipaamaa tietoa näkyviin suoraan ylläpitonäkymiin ja varmistaa, että moduulin aiheuttamat virheet ja varoitukset näkyvät suoraan ruudulla.

6.3.2 Moduulin asentaminen

Moduulin asentaminen aloitetaan moduulin tiedostojen kopioimisella hakemistoon /site/modules/, joko suoraan kyseiseen hakemistoon tai omaan hakemistoonsa sen alle. Oma hakemisto on hyödyllinen, mikäli moduuliin liittyy useampia tiedostoja, kuten esimerkiksi erillisiä aliluokkia, tyylitiedostoja, JavaScript-tiedostoja, tai kuvatiedostoja.

Mikäli moduulille luodaan oma hakemistonsa modules-hakemiston alle, se kannattaa nimetä moduulin mukaan. Suorituskykysyistä ProcessWire löytää itse asiassa automaattisesti vain ne moduulitiedostot, jotka ovat modules-hakemistossa, suoraan sen alla olevissa hakemistoissa, tai niiden alla olevissa ja moduulin nimen mukaan nimetyissä hakemistoissa:

- /site/modules/HelloWorld.module
- /site/modules/HelloWorld.module
- /site/modules/my-modules/HelloWorld.module
- /site/modules/my-modules/HelloWorld/HelloWorld.module

Alla näkyvät esimerkit sen sijaan eivät enää löydy automaattisesti, sillä hakemistorakenne on liian syvä tai oikeita nimeämiskäytäntöjä ei ole noudatettu:

- /site/modules/my-modules/my-module/HelloWorld.module
- /site/modules/my-modules/my-module/HelloWorld/HelloWorld.module

Kun moduuli on tuotu moduulihakemistoon, sen asentaminen onnistuu ylläpitonäkymästä: Modules-osiosta löytyy lista sivuston moduuleista, ja asentamattoman moduulin voi tältä listalta asentaa yhdellä klikkauksella. Mikäli uusi moduuli ei heti näy listalla, myös moduulivälimuistin tyhjennystä varten on moduulinäkymässä oma toimintonsa.

6.3.3 Esimerkkimoduulin sovelluskoodi kokonaisuudessaan

LISTAUS 37. on yksinkertaisen esimerkkimoduulimme vaatima sovelluskoodi kokonaisuudessaan. Tarkalleen ottaen tässäkin on jo hieman ylimääräistäkin, sillä demonstrointitarkoituksessa olemme lisänneet mukaan myös metodit `___install()` ja `___uninstall()`, joita emme varsinaisesti tässä yhteydessä tarvitse.

```
class ExampleModule extends WireData implements Module,
      ConfigurableModule {

    public static function getModuleInfo() {
        return array(
            'title' => "Example Module",
            'version' => "1.0.0",
            'author' => "Teppo Koivula",
            'summary' => "Yksinkertainen esimerkkimoduuli”,
            'singular' => true,
            'autoload' => true,
            'requires' => array('PHP>=5.4.1', 'ProcessWire>=2.4.1'),
        );
    }

    public static function getDefaults() {
        return array('selector' => 'template=basic-page');
    }

    public static function getModuleConfigInputfields(array $data) {
        $inputfields = new InputfieldWrapper();
        $data = array_merge(self::getDefaults(), $data);
        $input = wire('modules')->get('InputfieldSelector');
        $input->name = 'selector';
        $input->label = $this->_('Hakulauseke');
        $input->description = $this->_('Tämä hakulauseke määrittää '
                    . 'sivut, joiden tallennukseen moduuli reagoi.');
        $input->value = $data['selector'];
        $inputfields->add($input);
        return $inputfields;
    }
```

```
    public function init() {
        $this->addHookAfter("Pages::save", $this, "hookPagesSave");
    }

    public function hookPagesSave(HookEvent $event) {

        // otetaan Page-olio (eli sivu) talteen
        $page = $event->object;

        // verrataan sivua asetuksista löytyvään hakulausekkeeseen
        if ($page->is($this->selector)) {

            // asetetaan viesti, joka tulee näkyviin seuraavalla
            // ylläpitonäkymän sivulatauksella
            $this->session->message("Vastaava sivu tallennettu!");

            // lähetetään sähköpostitse tieto muutoksesta; jos sivun
            // kentässä to_field on arvo se määrittää vastaanottajan
            // (muutoin käytetään oletusosoitetta to@example.com)
            $to = $page->to_field ?: "to@example.com";
            $subject = "Sivu tallennettu: {$page->url}";
            $body = "{$user->name} tallensi sivun {$page->url} "
                  . date("j.n.Y H:i:s");
            wireMail($to, $from, $subject, $body);

        }

    }

    public function ___install() {
        $this->session->message("Suoritetaan asennusvaiheessa.");
    }

    public function ___uninstall() {
        $this->session->message("Suoritetaan poistovaiheessa.");
    }
}
```

LISTAUS 38. Esimerkinomainen moduulitiedosto, joka lähettää sähköpostia haluttuun osoitteeseen aina tietyt kriteerit täyttävien sivujen tallennuksen yhteydessä.

Seuraavissa luvuissa käsittelemme jokaista esimerkkimoduulin osaa erikseen, ja esittelemme samalla joitakin vaihtoehtoisia tapoja vastaavien toimintojen toteuttamiseen.

6.3.4 Valinnaiset asennusmetodit ___install() ja ___uninstall()

Metodien ___install() ja ___uninstall() avulla voidaan suorittaa sovelluskoodia joko moduulin asennuksen tai poiston yhteydessä. Tyypillisiä käyttötapauksia näille ovat esimerkiksi moduulin edellyttämien sivujen, oikeuksien, käyttäjäroolien, tai hakemistojen lisääminen moduulin asennusvaiheessa ja poistaminen moduulia poistettaessa.

Osa edellä mainituista toiminnoista – esimerkiksi sivun lisääminen asennuksen yhteydessä – onnistuu myös moduulin asetusten avulla, mutta asennus- ja poistometodit tarjoavat laajemmat mahdollisuudet komentojen suorittamiseen. Esimerkkimoduulissamme emme hyödynnä näitä, vaan ne ovat mukana vain demonstrointitarkoituksessa.

6.3.5 Moduulin perustiedot ja niiden välittäminen järjestelmälle

getModuleInfo()-metodi kertoo järjestelmälle, mistä moduulissa oikeastaan on kyse. Metodi palauttaa kutsuttaessa moduulin perustiedot, joihin lukeutuvat niin ihmisten luettaviksi tarkoitetut otsikko (title) ja kuvaus (summary), kuin myös kokoelma järjestelmän käyttöön tarkoitettuja, moduulin toimintaan vaikuttavia tietueita.

```
public static function getModuleInfo() {
    return array(
        'title' => "Example Module",
        'version' => "1.0.0",
        'author' => "Teppo Koivula",
        'summary' => "Yksinkertainen esimerkkimoduuli”,
        'singular' => true,
        'autoload' => true,
        'requires' => array('PHP>=5.4.1', 'ProcessWire>=2.4.1'),
    );
}
```

LISTAUS 39. Moduulin tiedot järjestelmälle välittävä osa esimerkkimoduulimme sovelluskoodista, getModuleInfo().

Staattinen metodi voidaan suorittaa luomatta luokasta ensin uutta oliota, mikä mahdollistaa moduulin tietojen tarkastelun moduulia asentamatta. Staattiset metodit eivät pääse kiinni luokan ei-staattisiin ominaisuuksiin tai -metodeihin, joten niiden tulee sisältää

palautettava data kokonaisuudessaan, kysellä se muilta staattisilta metodeilta, tai hyödyntää luokan staattisiksi määriteltyjä ominaisuuksia. (Zandstra 2008: 45-47.)

Versiosta 2.5 lähtien ProcessWire on mahdollistanut myös erillisen JSON- tai PHP-tiedoston hyödyntämisen edellä mainittujen perustietojen välittämiseen järjestelmälle: ModuleName.info.json tai ModuleName.info.php (Cramer 9/2014). Mikäli moduulin hakemistosta löytyy jompi kumpi edellä mainituista infotiedostoista, erillistä getModuleInfo()-metodia ei ole enää tarpeen toteuttaa.

Esimerkkimoduulimme hyödyntää vain murto-osaa kaikista moduulien käytössä olevista info-tietueista, jotka on kokonaisuudessaan listattu taulukkoon 4. Pakolliset tietueet on aina määriteltävä, mutta kaikki muut ovat vapaaehtoisia, ja voidaan siten jättää pois ellei niille ole erityistä tarvetta.

Asetus	Pakollinen	Kuvaus
title	Kyllä	Moduulin otsikko. Tätä tietuetta hyödynnetään sivuston ylläpitonäkymissä tarpeen mukaan.
version	Kyllä	Moduulin versio, joko kokonaislukuna (100 = 1.0.0, 101 = 1.0.1, 10 = 0.1.0) tai suoraan versionumerona (1.0.0, 1.0.1, 0.1.0)
summary	Kyllä	Moduulin kuvaus. Tätä tietuetta hyödynnetään pääasiassa sivuston ylläpitonäkymän moduuliosiossa.
singular	Vaihtelee	Onko moduulista mahdollista käynnistää vain yksi instanssi kerrallaan? Asetus on pakollinen, ellei moduuli sisällä erillistä isSingular()-metodia.
autoload	Vaihtelee	Halutaanko moduulin käynnistyvän aina järjestelmän mukana? Arvo voi olla true, false, tai valitsin – esimerkiksi template=admin, jolloin moduuli ladataan aina ylläpitonäkymissä. Asetus on pakollinen, ellei moduuli sisällä erillistä isAutoload()-metodia.

author	Ei	Moduulin tekijän nimi tai muu tunnistetieto.
href	Ei	Linkki lisätietoihin tai esimerkiksi tukisivustolle.
permanent	Ei	Onko moduuli pysyvä, eli ei käyttöliittymän kautta poistettavissa?
permission	Ei	Moduulin käytön edellyttämän oikeuden nimi.
permissions	Ei	Oikeudet, jotka lisätään automaattisesti moduulin asennuksen yhteydessä, ja poistetaan moduulin poistamisen yhteydessä.
requires	Ei	Moduulin riippuvuudet; esimerkiksi muut moduulit, PHP:n versio (PHP>=5.5.0), tai ProcessWiren versio (ProcessWire>=2.5.0)
installs	Ei	Muut moduulit, jotka tämän moduulin halutaan oman asennuksensa yhteydessä asentavan.
icon	Ei	Ikoni, joka näytetään tämän moduulin yhteydessä esimerkiksi ylläpitonäkymän valikoissa.

TAULUKKO 4. Listaus kaikista moduulikohtaisista info-tietueista.

6.3.6 Moduulikohtaiset konfigurointiasetukset

Edellä mainitut moduulin perustiedot ovat pysyviä, eivätkä muutu moduulin ajon aikana. Lisäksi käytettävissä olevat tietueet on määritelty jo etukäteen, eikä niihin voi sisällyttää omia, moduulikohtaisia asetuksia. Usein moduuleille on kuitenkin tarpeen määrittää omia asetuksiaan, lähtien esimerkiksi päivämäärien esitystavasta ja päätyen vaikkapa ulkoisten palveluiden autentikointitietoihin.

Luvussa 8 lyhyesti esitelty ConfigurableModule-rajapinta on olemassa juuri tällaisia moduulikohtaisia asetuksia varten. Sen toteuttavilla moduuleilla voi olla omia, kehittäjän määrittelemiä, ja sivuston ylläpitonäkymän kautta hallittavia asetuksiaan. Mainitun kaltaisten asetusten määrittelyyn on muutama vaihtoehtoinen menetelmä:

- Moduuli voi toteuttaa getModuleConfigInputFields()-metodin, joka palauttaa InputfieldWrapper-luokan instanssin. InputfieldWrapper on kokoelma lomakekenttiä, ja tässä tapauksessa kyseisen instanssin sisältämät lomakekentät määrittävät suoraan moduulin konfigurointiasetukset.
- Moduulitiedoston oheen voidaan lisätä ModuleNameConfig.php -tiedosto, joka määrittää ModuleNameConfig-luokan (ModuleName-alku korvataan moduulin oikealla nimellä). Kyseinen luokka periytyy ModuleConfig-pohjaluokasta, jolla puolestaan on moduulien konfigurointia varten tarvittavat metodit.
- Kolmantena ja kenties yksinkertaisimpana vaihtoehtona moduulitiedoston oheen voidaan lisätä ModuleName.config.php -tiedosto, jossa kaikki moduulikohtaiset konfigurointiasetukset määritetään alustamalla array-muuttuja $config.

Tässä opinnäytteessä hyödynnämme ensimmäistä yllä mainituista menetelmistä, eli getModuleConfigInputfields()-metodia. Tämä menetelmä takaa parhaan yhteensopivuuden myös vanhempien ProcessWire-versioiden suuntaan ja on myös kehittäjän kannalta varsin suoraviivainen ratkaisu. Listauksessa 40 on kuvattu kyseisen metodin toteutus esimerkkimoduulissamme.

```
public static function getDefaults() {
     return array('selector' => 'template=basic-page');
 }

 public static function getModuleConfigInputfields(array $data) {
     $inputfields = new InputfieldWrapper();
     $data = array_merge(self::getDefaults(), $data);
     $input = wire('modules')->get('InputfieldSelector');
     $input->name = 'selector';
     $input->label = $this->_('Hakulauseke');
     $input->description = $this->_('Tämä hakulauseke määrittää '
                . 'sivut, joiden tallennukseen moduuli reagoi.');
     $input->value = $data['selector'];
     $inputfields->add($input);
     return $inputfields;
 }
```

LISTAUS 40. Konfiguraatioasetukset muodostava osa esimerkkimoduulimme sovelluskoodista, metodit getDefaults() ja getModuleConfigInputfields().

Staattinen metodi getDefaults() yksinkertaisesti palauttaa oletusasetukset, joita käytetään, mikäli moduulia ei vielä ole konfiguroitu käyttäjän toimesta. Toinen staattinen metodi, getModuleConfigInputfields(), saa parametrinaan mahdolliset ennalta täytetyt konfiguraatioasetukset, yhdistää ne getDefaults()-metodin palauttamiin oletusarvoihin, ja luo sekä palauttaa moduulin konfigurointiin tarvittavat lomakekentät.

Konfigurointiin käytettävillä lomakekentillä on tyypillisesti ominaisuudet "name", "label", "description", "notes", ja "value". Ainoa esimerkissämme käytettävä konfiguraatiokenttä on tyypiltään InputfieldSelector ja mahdollistaa valitsimen luonnin visuaalisen käyttöliittymän avulla. Muita mahdollisia tyyppejä ovat esimerkiksi InputfieldText, InputfieldTextarea, InputfieldSelect, ja InputfieldCheckbox.

Konfigurointikenttiä voidaan määritellä niin monta kuin moduuli konfiguraatioasetuksia tarvitsee, ja niiden arvoja voidaan lopulta hyödyntää suoraan moduulin sovelluskoodissa. Esimerkkinä käyttämämme kentän tapaan muukint kentät haetaan wire('modules')->get(...) -kutsulla, konfiguroidaan, ja lopulta lisätään $inputfields-olioon.

6.3.7 Moduulin alustusmetodit __construct(), init(), ja ready()

__construct() on PHP:n alustusmetodi, joka suoritetaan aina automaattisesti heti kun sen sisältävästä luokasta luodaan uusi olio. Alustusmetodin tärkein tehtävä on yleensä olion ominaisuuksien alustaminen, ja sille voidaan myös antaa parametreja olion luonnin yhteydessä: new exampleModule($arg1, $arg2, ...). (Tatroe, MacIntyre, ja Lerdorf 2013: 161-162; Zandstra 2008: 23-24.)

PHP:n natiivin alustusmetodin suorituksen yhteydessä ProcessWiren tarjoama ohjelmointirajapinta (API) ei ole vielä kokonaisuudessaan käytettävissä. Erityisen hyödylliseksi sen kuitenkin tekevät paitsi kyky vastaanottaa parametreja ja asettaa oletusarvoja olion ominaisuuksille, myös mahdollisuus tehtävien suorittamiseen mahdollisimman varhaisessa vaiheessa olion (ja järjestelmän) elinkaarta.

init() puolestaan on ProcessWiren oma alustusmetodi, jonka moduulit voivat tarvittaessa toteuttaa. Mikäli kyseinen metodi on määritelty, se suoritetaan heti kun moduuli on ladattu ja sen konfiguraatioasetukset on valmisteltu käyttöön. Koska autoload-moduulit

ladataan automaattisesti järjestelmän mukana, niiden osalta init() suoritetaan järjestelmän käynnistämisen yhteydessä.

ready() on toinen ProcessWiren omista alustusmetodeista, ja suoritetaan myöskin automaattisesti moduulin latauksen yhteydessä, mutta vain autoload-moduulien osalta. Koska ready() suoritetaan init()-metodin jälkeen, ja vasta kun ohjelmointirajapinta (API) on kokonaisuudessaan käytettävissä, tässä vaiheessa voidaan esimerkiksi suorittaa aktiiviseen sivuun liittyviä tehtäviä (tieto aktiivisesta sivusta ei vielä init()-metodin suorituksen aikana ole käytettävissä).

Mikäli moduulin on tarkoitus liittää koukkuja, tämä tehdään useimmiten joko init()- tai ready()-metodeissa. Esimerkkimoduulissamme koukku kiinnitetään init()-metodissa hyödyntäen addHookAfter()-metodia. Tämä metodi löytyy nykyiseltä oliolta ($this) koska kyseinen olio periytyy WireData-luokasta, joka puolestaan periytyy Wire-luokasta, joka toteuttaa kyseisen metodin.

```
public function init() {
    $this->addHookAfter("Pages::save", $this, "hookPagesSave");
}
```

LISTAUS 41. Koukut kiinnittävä osa esimerkkimoduulimme sovelluskoodista, init().

6.3.8 Koukkujen kiinnittäminen

Koukkujen kiinnittäminen onnistuu edellisen init()-esimerkin tapaan kutsumalla WireData-oliolta perittyjä, tähän käyttöön tarkoitettuja metodeja. Mainittuja metodeja on yhteensä neljä kappaletta, ja ne mahdollistavat koukkujen kiinnittämisen ennen, jälkeen, tai korvaamaan halutun koukkuja tukevan metodin.

Koukkuja tukevat metodit tunnistaa pääsääntöisesti metodin nimen alusta löytyvistä kolmesta alaviivasta: public function ___save() { .. }. Tämä on järjestelmän sisäänrakennettu ominaisuus, ja käytännössä mikä tahansa metodi, joko ydinkoodissa tai omissa moduuleissa, voidaan muuttaa koukkuja tukevaksi lisäämällä edellä mainitut alaviivat sen nimen alkuun.

- addHook($method, $toObject, $toMethod=null, $options=array()) on aina kaikkien muiden koukkujen liittämiseen käytettävien metodien taustalla, ja ottaa vastaan laajan joukon parametreja, joista tärkeimmät ovat $method (kohdemetodin nimi, esimerkiksi Pages::save), $toObject (olio, josta koukkumetodi haetaan), sekä $toMethod (koukkumetodin nimi). Tiedosto/wire/core/Wire.php sisältää tarkemman kuvauksen käytössä olevista asetuksista ($options-parametri).
- addHookBefore($method, $toObject, $toMethod=null, $options=array()) välittää parametrinsa suoraan addHook()-metodille, mutta kertoo oletuksena sille $options-parametriä hyödyntäen, että koukkumetodi halutaan suorittaa ennen kohdemetodia.
- addHookAfter($method, $toObject, $toMethod=null, $options=array()) vastaa muutoin addHookBefore()-metodia, mutta koukkumetodi suoritetaankin vasta kohdemetodin jälkeen.
- addHookProperty($property, $toObject, $toMethod=null, $options=array()) lisää uuden ominaisuuden ("property") kohteena olevaan olioon. Myös tämä metodi on käytännössä vain lyhyempi tapa kutsua addHookMethod()-metodia tietyillä, valmiiksi määritellyillä $options-parametreilla.

Koukkumetodin nimi (esimerkissämme hookPagesSave) on vapaasti määriteltävissä, mutta sen tulee löytyä määritetyn kohdeolion sisältä (esimerkissämme $this, eli nykyinen olio). Mikäli koukku kiinnitettäisiin kokonaan luokan ulkopuolella, esimerkiksi aihiotiedostossa, tämä parametri jätettäisiin tyhjäksi (null) ja koukkumetodi määriteltäisiin aihiotiedostossa uutena funktiona.

Eräs ProcessWiren koukkujen erityispiirteistä on se, että niillä voidaan paitsi tarttua koukkuja tukeviin metodeihin, myös lisätä uusia metodeja ja ominaisuuksia valmiisiin ydinkomponentteihin tai moduuleihin. Uuden ominaisuuden lisäämiseen löytyy valmis komento addHookProperty() ja uuden metodin lisääminen onnistuu yksinkertaisesti antamalla parametriksi kohdemetodi, jota ei vielä kyseisessä luokassa ole olemassa.

6.3.9 Koukkumetodi hookPagesSave()

Siinä missä kaikki tähän asti esitelty on ollut oikeastaan tukitoiminnallisuutta, viimeinen ja tärkein osio moduulistamme on koukkumetodi hookPagesSave(), joka vastaa moduulin varsinaisesta toiminnallisesta osuudesta. Tässä kohtaa lienee kuitenkin syytä mainita,

että esimerkkimoduulimme sisältää likipitäen kaikki moduulilta koskaan vaaditut ominaisuudet, mikä kasvattaa "tukitoimintojen" määrää merkittävästi.

Tämän moduulin tapauksessa pakollisia toimintoja olisivat oikeastaan olleet vain metodit getModuleInfo(), init(), ja hookPagesSave(). Minimitoteutuksella moduulimme lähdekoodi olisi siis käytännössä puolittunut. Vaikka osa moduuleista onkin kokonaisia, laajamittaisia sovelluksia, pienen ja erikoistuneen moduulin voi toteuttaa jopa alle kymmenen koodirivin, mikä kertoo omalta osaltaan jotain ProcessWiren moduulitoteutuksen joustavuudesta.

```
    public function hookPagesSave(HookEvent $event) {

        // otetaan Page-olio (eli sivu) talteen
        $page = $event->object;

        // verrataan sivua asetuksista löytyvään hakulausekkeeseen
        if ($page->is($this->selector)) {

            // asetetaan viesti, joka tulee näkyviin seuraavalla
            // ylläpitonäkymän sivulatauksella
            $this->session->message("Vastaava sivu tallennettu!");

            // lähetetään sähköpostitse tieto muutoksesta; jos sivun
            // kentässä to_field on arvo se määrittää vastaanottajan
            // (muutoin käytetään oletusosoitetta to@example.com)
            $to = $page->to_field ?: "to@example.com";
            $from = "from@example.com";
            $subject = "Sivu tallennettu: {$page->url}";
            $body = "{$user->name} tallensi sivun {$page->url} "
                  . date("j.n.Y H:i:s");
            wireMail($to, $from, $subject, $body);

        }

    }
```

LISTAUS 42. Koukkumetodin hookPagesSave() toteutus esimerkkimoduulissamme.

Alustusmetodissa init() ohjeistimme järjestelmää suorittamaan koukkumetodin hookPagesSave() aina sivun tallennuksen jälkeen komennolla $this->addHookAfter("Pages::save", $this, "hookPagesSave"). Kyseinen koukkumetodi

vertaa tallennettua sivua asetuksistaan löytyvään valitsimeen, ja valitsimen osuessa kohdalleen tiedottaa ylläpitäjää sähköpostitse muutoksesta.

Sähköpostin lähettämiseen hyödynnetään wireMail()-funktiota, joka on yksi ProcessWiren /wire/core/Functions.php -tiedostosta löytyvistä API-funktioista. Olisimme toki voineet hyödyntää myös suoraan PHP:n mail()-funktiota, mutta wireMail() on suositeltu tapa, koska se mahdollistaa viestin lähettämisen järjestelmään mahdollisesti asennettujen WireMail-tyyppisten moduulien avustuksella.

Yllä näkyvä lähdekoodi on kommentoitu, ja kommenteista selviää melko tarkasti kunkin koodin osan tehtävä. Koukkumetodi saa aina parametrikseen HookEvent-luokkaa edustavan olion, jonka ominaisuus "object" tässä tapauksessa sisältää käsiteltävän sivun. HookEvent-oliota käsitellään seuraavassa osiossa vielä hieman tarkemmin.

6.3.10 Koukkuolio HookEvent

Jokainen koukkumetodi saa parametrinaan luokan HookEvent instanssin, johon usein viitataan muuttujanimellä $event. HookEvent-luokalla on kokoelma ominaisuuksia ja metodeja, joista olennaisimmat on listattu alle.

- $event->object sisältää viittauksen siihen olioon, josta koukkuolio on peräisin; mikäli koukku kiinnitetään kohteeseen Pages::save, $event->object on viittaus tallennettuun sivuun
- $event->method sisältää lähdemetodin nimen – joka siis on se metodi, johon koukkumetodi on kiinnitetty, eli esimerkiksi jos koukun kohteena on Pages::save, $event->method saa arvokseen "save"
- $event->arguments on numeerinen array, joka sisältää lähdemetodin parametrit; jos koukkumetodi liitetään sivun tallennukseen (Pages::save) ja sivu tallennetaan parametrilla ”title” ($pages->save('title')), $event->arguments(0) on "title"
- $event->arguments() huolii parametrikseen numeron tai nimen ja palauttaa lähdemetodilta kyseistä numeerista sijaintia tai nimeä vastaavan parametrin; toisin sanoen arguments() on kuten ominaisuus "arguments”, vain metodin muodossa
- $event->return pätee after-koukkuihin – sekä before-koukkuihin jos parametrin "replace" arvoksi on asetettu true – ja sisältää lähdemetodin paluuarvon; pa-

luuarvon muokkaaminen vaikuttaa suoraan lähdemetodin paluuarvoon, ja mahdollistaa näin epäsuorasti metodin toimintaan vaikuttamisen

- `$event->replace` on parametri, jonka asettaminen arvoon `true` before-koukussa aiheuttaa sen, että koukkumetodi korvaa kokonaisuudessaan alkuperäisen metodin; tätä tapaa vaikuttaa järjestelmän suoritukseen ei yleensä suositella, koska se on erityisen riskialtis ja vaatii yhteensopivan toiminnon toteuttamista kokonaan tyhjältä pöydältä

Muita koukkuolion ominaisuuksia ovat "options", "data", "id", ja "when", mutta nämä liittyvät yleensä monimutkaisempiin käyttötapauksiin, emmekä tässä yhteydessä paneudu niihin sen enempää. Yllä näkyvät ominaisuudet kattavat jo selkeästi suurimman osan koukkujen käyttötapauksista.

6.4 Yhteenveto ja pohdinta

ProcessWiren moduulikehitys on tietyllä tapaa välimuoto sovelluskehityksen parhaiden käytäntöjen ja helpon ymmärrettävyyden väliltä. Tietyissä tilanteissa sitä on kritisoitu esimerkiksi staattisten metodien hyödyntämisestä, jota moni kehittäjä pitää merkkinä huonosta ohjelmointitavasta, mutta toisaalta juuri näiden "joustojen" avulla on saatu aikaan yksinkertaistuksia, jotka mahdollistavat moduulikehityksen aloittamisen myös ilman vankkaa ohjelmointitaustaa.

Moduulien kehittämisen alkeet oppii helpoiten käytännön kautta, eli toteuttamalla omia moduuleja. Yksinkertaisiin tarkoituksiin suunnattujen moduulien laatiminen on lopultakin hyvin helppoa, ja juuri tässä piilee ProcessWiren moduulitoteutuksen hienous: aloittaminen on helppoa ja pienellä vaivalla saa paljon aikaan, mutta toisaalta järjestelmä tekee mahdolliseksi myös erittäin laajojen, sovellusmaisten moduulien toteuttamisen.

Aloittelevan kehittäjän kannattaa tutustua myös valmiisiin moduuleihin, joita ProcessWiren moduulikirjastosta löytyy satoja. Luku ei ole erityisen vaikuttava esimerkiksi WordPressin kymmenien tuhansien lisäosien rinnalla, mutta kuitenkin niin laaja, että hyviä esimerkkejä löytyy taatusti. Lisäksi ProcessWiren moduulikirjastoon lisätyt moduulit ovat kevyesti auditoituja, joten niiden laatu on keskimäärin varsin hyvällä tasolla.

LYHENTEET JA SANASTO

Lyhenteet ja vieraskieliset termit

CMS, WCMS	Content Management System eli sisällönhallintajärjestelmä on sovellus, jonka avulla on mahdollista luoda, muokata, ja organisoida sisältöä. Tämän opinnäytteen kontekstissa termillä viitataan pääasiassa verkkosisältöjen hallinnointiin käytettyihin järjestelmiin eli verkkosisällönhallintajärjestelmiin (Web Content Management System).
WAF	Web Application Framework eli web-sovelluskehys on yleisluontoinen, monikäyttöinen, verkkopohjaisten ratkaisujen kehittämiseen suunniteltu sovellusalusta. Tässä opinnäytteessä sovelluskehyksistä puhuttaessa viitataan ensisijaisesti juuri web-sovelluskehyksiin.
CMF	Content Management Framework eli sisällönhallintakehys muistuttaa sovelluskehystä, mutta tarjoaa sisällönhallintajärjestelmän tyyppisiä valmiita ratkaisuja. Osa sisällönhallintajärjestelmistä voidaan ominaisuuksiensa puolesta luokitella myös sisällönhallintakehyksiksi.
API	Application Programming Interface eli ohjelmointirajapinta määrittää tavan, jolla muut sovellukset voivat kommunikoida, vaihtaa tietoja ja mahdollisesti jopa käskyttää tiettyä sovellusta tai palvelua.
DRY	Don't Repeat Yourself. Sovelluskehityksen periaate, jonka mukaan minkä tahansa identtisen sisällön toistamista useammassa kuin yhdessä paikassa tulisi välttää.
MVC	Model-View-Controller on sovelluskehityksen arkkitehtuurimalli, jonka lähtökohtana on sovelluksen jakaminen itsenäisiin, toisiaan täydentäviin komponentteihin.
WYSIWYG	What You See Is What You Get. Työkalu, jossa muokattava sisältö mukailee jo muokkausvaiheessa lopullista esitysmuotoaan, ja jossa tyypillisesti hyödynnetään esimerkiksi tekstinkäsittelystä tuttuja muokkaustyökaluja.

CDN	Content Distribution Network, myös Content Delivery Network. Palvelu, joka mahdollistaa alkuperäisen sisällön hajauttamisen erillisiin Internet-sijainteihin, tavoitteena kuorman tasaaminen ja ja palvelun nopeuden parantaminen.
AJAX	Asynchronous JavaScript and XML. Yleisnimitys tavalle toteuttaa asynkronisia pyyntöjä JavaScript-kielellä muun muassa XMLHttpRequest-oliota hyödyntäen.
DDoS	Distributed Denial of Service. Palvelunestohyökkäys, jossa palvelimeen tai palveluun kohdistetaan useista lähdeosoitteista suuri määrä liikennettä, tavoitteena estää sen tarkoituksenmukainen toiminta.

Yleisesti ohjelmistokehitykseen liittyvät käsitteet

Järjestelmäintegraatio	Järjestelmäintegraatiolla tarkoitetaan erillisten järjestelmien yhdistämistä siten, että ne hyödyntävät toistensa tietoja ja toimivat tarpeellisilta osin yhtenäisenä kokonaisuutena.
Avoin lähdekoodi	Avoimella lähdekoodilla (open source) viitataan yleisesti sovelluksiin, joita saa käyttää, muokata, ja jakaa eteenpäin rajoituksetta. Avoin lähdekoodi ja vapaa ohjelmisto ovat sisarkäsitteitä, joiden erot ovat osittain filosofisia.
Tietoturva	Tietoturva käsittää tietojen, järjestelmien, ja palveluiden toimimisen siten, että niissä oleva tieto on vain erikseen määriteltyjen asianomaisten saavutettavissa ja muokattavissa, eli suojassa tietomurroilta sekä ilkivallalta.
Välimuisti	Välimuisti (cache) on tapa tallentaa valmiiksi noudettua ja käsiteltyä tietoa sellaiseen muotoon, josta se on myöhemmin otettavissa käyttöön aiempaa helpommin, nopeammin, ja tehokkaammin.

ProcessWire-järjestelmän tärkeimmät käsitteet

Sivupohja	Sivupohjat (templates) ovat järjestelmän tietotyyppejä. Kukin sivupohja on kokoelma kenttiä, asetuksia, ja metatietoa.

Aihiotiedosto	Aihiotiedosto (template file) on sivupohjan näkyvä ilmentymä. Sivupohja määrittää tietorakenteen ja aihiotiedosto määrittää sen, miten sivupohja esitetään sivuston selaajille.
Kenttätyyppi	Kenttätyyppi (fieldtype) määrittävät kentän tietorakenteen, sekä sen, miten tietokantaan tallennettavia tai sieltä noudettavia rivejä käsitellään, sekä mitä lomakekenttiä kentän sisällön muokkaukseen voidaan hyödyntää.
Kenttä	Kenttä (field) on tietyn kenttätyypin nimetty, yksilöity ilmentymä. Kukin kenttä hyödyntää tiettyä, kenttätyypin kanssa yhteensopivaa lomakekenttää ja voi kuulua yhteen tai useampaan sivupohjaan.
Lomakekenttä	Lomakekenttä (inputfield) on tietyn kentän sivuston hallintanäkymissä näkyvä ilmentymä. Lomakekentät mahdollistavat kenttiin tallennettujen tietojen muokkauksen järjestelmän käyttöliittymän kautta.
Bootstrap-menetelmä	Bootstrap-menetelmä viittaa ProcessWiren tapauksessa tapaan jolla järjestelmä voidaan ohjelmallisesti käynnistää toisen PHP-kielellä kirjoitetun ohjelman sisään suorittamalla sen käynnistyksestä vastaava tiedosto.
Valitsin	Valitsimet (selectors) mahdollistavat sivujen ja muiden sisältöolioiden noutamisen valitsimen määrittämien hakuehtojen perusteella. ProcessWiren valitsimet muistuttavat jQueryn valitsimia, jotka puolestaan pohjautuvat CSS:n valitsimiin.
Moduuli	ProcessWiren moduulit ovat luokkia, joihin on paketoitu tietty toiminto tai kokoelma toimintoja. Kullakin moduulityypillä on järjestelmän toiminnassa oma roolinsa.
Koukku	ProcessWiren koukut (hooks) mahdollistavat kehittäjän määrittelemän sovelluskoodin kiinnittämisen järjestelmän omiin metodeihin. Koukkumetodi voidaan suorittaa joko ennen tai jälkeen järjestelmän oman metodin, se voi korvata alkuperäisen metodin kokonaisuudessaan, tai se voi lisätä järjestelmän olioille kokonaan uusia toimintoja tai tietueita.

LÄHTEET

Abela, R. 2014. WordPress Vulnerabilities Statistics. Luettu 7.10.2015. http://www.wpwhitesecurity.com/wordpress-security/statistics-highlight-main-source-wordpress-vulnerabilities/

Coggeshall, J. ja Tocker, M. 2009. Zend Enterprise PHP Patterns. Apress.

Cramer, R. 2012. Introducing ProcessWire 2.2. Luettu 12.10.2015. https://processwire.com/about/news/introducing-processwire-2.2/

Cramer, R. 2/2014. Introducing ProcessWire 2.4. Luettu 7.10.2015. https://processwire.com/about/news/introducing-processwire-2.4/

Cramer, R. 9/2014. ProcessWire 2.4 Changelog. Luettu 7.10.2015. https://processwire.com/blog/posts/processwire-2.5-changelog/

Cramer, R. 2015a. About ProcessWire ProCache. Luettu 7.10.2015. https://processwire.com/api/modules/procache/

Cramer, R. 2015b. How to structure your template files: Direct Output. Luettu 7.10.2015. https://processwire.com/docs/tutorials/how-to-structure-your-template-files/page2

Herrmann, M. 2015. Wireshell - processwire command-line companion. http://wireshell.pw/

Kaspersky Lab 2014. Global IT Security Risks Survey 2014 – Distributed Denial of Service (DDoS) Attacks. Luettu 7.10.2015. https://press.kaspersky.com/files/2014/11/B2B-International-2014-Survey-DDoS-Summary-Report.pdf

Knoll, N. 2015. ProcessWire Module Generator. Luettu 7.10.2015. http://modules.pw/

Reenskaug, T. 5/1979. Thing-Model-View-Editor: an Example from a planningsystem. http://heim.ifi.uio.no/~trygver/1979/mvc-1/1979-05-MVC.pdf

Reenskaug, T. 12/1979. Models - Views - Controllers. Luettu 7.10.2015. http://heim.ifi.uio.no/~trygver/1979/mvc-2/1979-12-MVC.pdf

Reid, M. 2014. MODX Revolution: Resources. Luettu 7.10.2015. http://rtfm.modx.com/revolution/2.x/making-sites-with-modx/structuring-your-site/resources

Shiflett, C. 2006. Essential PHP Security. O'Reilly Media, Inc.

Tatroe, K., MacIntyre, P. ja Lerdorf, R. 2013. Programming PHP. O'Reilly Media, Inc.

Wessels, D. 2001. Web Caching. O'Reilly Media, Inc.

Zalewski, M. 2012. The Tangled Web: a Guide to Securing Modern Web Applications. No Starch Press, Inc.

Zandstra, M. 2008. PHP Objects, Patterns and Practice. Apress.